UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Solução Cloud Based Development - CRM

João Pedro Costa Ramos dos Santos

Mestrado em Engenharia Informática
Especialização em Sistemas de Informação

Trabalho de Projeto Orientado Por:

Professora Doutora Helena Isabel Aidos Lopes Tomás

2019

Agradecimentos

A toda a minha família, em particular aos meus pais, irmã e avós. E a todos os amigos.

Dedicatória.

Resumo

"(…) CRM is not just about installing software or automating customer touchpoints. It is about the reinvention of our enterprises around the customer. It is about becoming and remaining customer-centric." - *Francis Bottle, Customer Relationship Management [1]*

Não é nova a utilização do acrónimo *CRM* que, por extenso, se traduz para "*Customer Relationship Management*". Qualquer negócio ou empresa tem clientes, e com clientes surge a necessidade de facilitar as relações cliente-empresa e empresa-cliente. Deste modo, também não é nova a necessidade de soluções *CRM* para estas entidades.

O presente relatório refere-se ao projeto de estágio, realizado em nome da Accenture, para um cliente do ramo dos serviços postais. O projeto consiste na implementação de um sistema *CRM* central, capaz controlar os múltiplos setores de negócio do cliente por integração com os seus diversos sistemas e serviços, e que facilite a receção, tratamento, gestão e análise dos dados relativos às relações com clientes. Este sistema *CRM* inclui as componentes *Sales* e *Service*, no entanto, será a segunda o foco deste trabalho. Para tudo isto, foi implementada uma solução *CRM cloud based* utilizando o *Engagement Cloud* da *Oracle* e expandida com uma aplicação externa à medida do negócio. O trabalho divide-se, por isso, em três vertentes principais, o trabalho no *Oracle Engagement Cloud (OEC)*, o trabalho externo ao *Oracle Engagement Cloud* e o trabalho no *Oracle Business Intelligence (OBI)*. Sendo que todas estas vertentes têm por base o *OEC*. Esta utilização do *OEC* como suporte aplicacional leva-nos a concluir que esta aplicação *cloud* não está preparada para o grau de complexidade do negócio deste cliente. Isto reflete-se essencialmente na necessidade de inclusão direta de aplicações externas no *OEC*, que, não obstante, integram de forma limpa e totalmente funcional.

Palavras-chave: *CRM, Engagement Cloud, Cloud, Orable BI*

Abstract

"(...) CRM is not just about installing software or automating customer touchpoints. It is about the reinvention of our enterprises around the customer. It is about becoming and remaining customer-centric." - Francis Buttle, Customer Relationship Management [1]

It is not new the use of the acronym CRM, which translates into, "Customer Relationship Management". Any business or company has clients, and with clients comes the need to improve and facilitate the relationships between company and client. Thus, it is also not new the need for CRM solutions for these entities.

This report refers to the internship project, developed with Accenture, for a client in the field of postal services. The project consists in the implementation of a central CRM system, capable of controlling the client's multiple sectors of business through the integration with their several systems and services, and which also facilitates the reception, treatment, management and analysis of data relative to their clients. This CRM system includes the Sales and Service components, however, only the latter will be addressed in detail during this report. For all of this, a CRM cloud-based solution was implemented using Engagement Cloud from Oracle and expanded with a tailored external application. Therefore, the work is divided in three major sections, the work within Oracle Engagement Cloud (OEC), the work outside Oracle Engagement Cloud and the work in Oracle Business Intelligence (OBI). All these sections have OEC as their baseline. This usage of OEC as a support application leads us to the conclusion that this cloud application is not suited for the degree of complexity of this client's business rules. This reflects mainly through the necessity to include external applications directly inside OEC, however, this process is handled smoothly and efficiently by the cloud application.

Keywords: CRM, Engagement Cloud, Cloud, Oracle BI

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo serve de apresentação ao conceito de *Customer Relationship Management*, contextualizando-o na situação de negócio atual. Serão abordados os problemas que esta solução tipicamente resolve, os principais objetivos deste projeto e ainda uma visão global sobre a organização deste documento.

Durante muitos anos, os negócios centravam-se no seu produto, e a atenção dada ao cliente tinha por base dados estatísticos e demográficos genéricos. No *CRM* o cliente torna-se o centro do negócio, sendo que o conceito engloba qualquer prática, estratégia ou tecnologia desenvolvida para melhorar as relações do negócio para o cliente. No mercado atual, o *CRM* está intimamente associado a aplicações web ou software. No entanto, apenas uma vertente do *CRM* é realmente relevante no contexto contemporâneo, o *CRM* na *cloud*. A *cloud* foi a maior alteração aplicada nos sistemas *CRM* desde o seu aparecimento. A *cloud* é um conceito que descreve uma rede de máquinas de armazenamento e processamento de informação. Esta informação pode ser acedida em qualquer local, através de um qualquer dispositivo com acesso à internet e capaz de suportar uma aplicação *web*. Dentro da variante do *CRM* na *cloud* existem duas grandes opções de implementação, o *Salesforce* e o *Engagement Cloud* da *Oracle*.

Um negócio só existe se tiver um mercado para se inserir, ou seja, se existirem clientes alvo para o produto oferecido. Assim, para vender um produto ou serviço é essencial existirem boas condições de ligação entre clientes e negócios. Este é então o problema que o *CRM* se propõe resolver. Para muitos, e para o contexto do presente documento, o acrónimo *CRM* significa "*Customer Relationship Management*", no entanto, existe quem traduza esta abreviatura de três letras para "*Customer Relationship Marketing*". Existem até, ainda que mais incomuns, referências ao conceito em que o termo "*Customer*" é omitido, ou ainda outras em que o termo "*Relationship*" é retirado. Seja qual for a situação, o conceito *CRM* está nitidamente associado a práticas centradas no cliente. Esta disparidade de definições deve-se ao facto de o conceito em si ser relativamente recente e amplo. Atualmente, e apesar do mercado estar povoado de

soluções e aplicações *CRM*, este conceito tem a sua origem "oficial" apenas em 1993 aquando da fundação da empresa *Siebel Systems Inc*. por parte de Tom Siebel. Mas ainda assim, a ideia em si existe já desde o início dos anos setenta, quando a satisfação dos clientes era avaliada através de inquéritos anuais. Nessa altura, os negócios estavam dependentes de "*Mainframes*" independentes (computadores dedicados a processamento) para automatizar os seus sistemas de vendas, sendo que, algumas mais avançadas chegavam a permitir a organização dos clientes em listas. A partir da década de oitenta alterações bruscas na forma de fazer negócio levaram a uma alteração no poder do cliente. Uma vez que existia mais oferta que procura as empresas tinham pouca margem de manobra nos preços praticados, sendo então a relação com os clientes a maior segurança que uma empresa podia ter para sobreviver face à sua competição.

Na perspetiva de empresas de tecnologia, que desenvolvem e aplicam estes sistemas, o *CRM* simboliza qualquer aplicação que automatize os processos de marketing, vendas e serviços de um determinado negócio. Estes sistemas apresentam ainda uma estrutura típica, de três níveis, comummente apelidados de "Níveis do *CRM*", e que podem ser consultados na tabela abaixo (Tabela 1).

Nível do CRM	**Descrição**
Estratégia	Perspetiva no *CRM* que olha para o negócio como algo centrado no cliente. Tendo como objetivo a aquisição e manutenção de clientes rentáveis.
Operacional	Perspetiva no *CRM* que se foca na automação dos projetos, mais concretamente a automação de Vendas, Serviços e Marketing.
Analítico	Perspetiva no *CRM* focada na análise inteligente de dados do cliente com fins estratégicos para o negócio.

Tabela 1. Níveis do CRM

1.1 Motivação

Na situação do projeto atual, o objetivo passa por implementar um sistema de *CRM* para uma empresa de serviços postais. O interesse principal está em adquirir maior conhecimento sobre o cliente, simplificar e uniformizar os processos de negócio e desenvolver uma visão de controlo de gestão. Sobre isto, o projeto tem várias visões.

A visão 360º sobre o cliente, que possibilite às equipas comerciais de apoio ao cliente e de marketing uma visão global de todas as interações do cliente com a empresa nos seus diferentes canais de contacto, componentes de Vendas, Serviços e *Marketing*.

A integração e visão *E2E* (*End to End*) de processos, que visa dar às equipas um acesso facilitado a informação integrada dos processos de suporte relativos à sua atividade e ao negócio.

Os relatórios analíticos, que visam proporcionar às equipas, funcionalidades de "*reporting*" e extração de informação analítica de gestão e de suporte à atividade, nas três componentes do *CRM*.

De um modo geral, a motivação deste projeto passa por preparar um sistema customizado às necessidades do cliente, de forma a que os seus processos de negócio sejam simplificados e unificados numa só plataforma. Esta simplificação vai desde a automação dos processos, tanto de comunicação como de análise, até à integração de todos os sistemas individuais num ambiente único.

O projeto passou por uma fase inicial, e anterior ao presente relatório, de desenvolvimento da componente de Vendas do *CRM*. Essa componente estava já implementada aquando da integração no projeto. Em seguida o projeto virou-se, então, para a componente de Serviços, que é a componente central para os trabalhos desenvolvidos e descritos neste documento.

1.2 Objetivos do Projeto

Os objetivos de um projeto *CRM* tendem a ser semelhantes para qualquer empresa que o empregue no seu negócio. No entanto, nem todos os negócios precisam do *CRM* em toda a sua extensão. Em seguida, serão descritos os principais objetivos da empresa para a qual o projeto está a ser desenvolvido.

O primeiro objetivo do *CRM* para este projeto está em melhorar o conhecimento sobre o cliente. Para isto foi desenvolvida uma visão única do cliente, transversal à organização e com dados atualizados. Será melhorada a experiência do cliente e serão registadas todas as interações dos clientes com a empresa, independentemente do canal de entrada. Uma vez que a empresa tem um negócio abrangente, e diversos canais de interação, torna-se essencial uma unificação dos processos de interação com os clientes.

Simplificar os processos e implementar ferramentas ágeis. Isto passa por dotar a força de vendas com uma ferramenta que permita melhorar o desempenho da atividade comercial. Além disso, é importante permitir uma análise de ofertas e *pricing* suportada por ferramentas analíticas. Este objetivo do negócio foi alcançado anteriormente na

implementação da componente de vendas do *CRM*, deste modo, não existe necessidade de o aprofundar mais.

De seguida, é necessário aumentar a eficácia comercial da empresa, algo que envolve a implementação de algumas medidas. Melhoria e potenciamento de uma comunicação proactiva com o cliente. Estabelecimento de consistência e integração do processo de venda. Medição do desempenho da força de vendas. Definir fluxos de trabalho para elaboração e aprovação de propostas. Realizar segmentação refinada do cliente através da recolha e tratamento de dados. Novamente, estes objetivos foram já alcançados na implementação do segmento de vendas.

Por fim, aumentar eficiência dos serviços, que será por onde vai passar a maior fatia do presente relatório. Aqui pretende-se, fornecer uma visão 360º do cliente em *front-office* através da disponibilização de informação pertinente e atualizada. Definir fluxos de trabalho eficientes para resolução de cada tipo de contacto. E simplificar e uniformizar as aplicações de suporte ao cliente e apoio ao negócio.

Para este projeto e consequente relatório o âmbito central passa pela componente dos serviços, isto porque, novamente, a componente de vendas já se encontra publicada. Para os serviços pretende-se ter tudo o que é necessário para registar e visualizar na gestão do cliente, em particular em questões de pós-venda. Isto é, o registo de todas as interações das diversas ramificações da empresa mãe com os seus clientes, por forma a garantir a sua satisfação. Para isto teremos, Registo de Interações, Registo e Tratamento de Solicitações/Pedidos, Incidências Operacionais, Pedidos de recolha, segundas entregas ou próprio dia, Serviços de tratamento de Solicitações/Pedidos, Visão de Cliente, *Reporting* e *Analytics* e *Knowledge Base*.

A última fase da componente operacional do *CRM*, o *Marketing*, ainda não tem objetivos definidos, à data da redação deste documento.

1.3 Objetivos Individuais

Falando dos objetivos específicos para mim estes passam pela implementação dos objetivos do projeto para a componente de *"Service"*, ressalvando que estes eram os objetivos à entrada no projeto, pelo que, podem não se verificar na totalidade. Em traços gerais isto traduz-se no seguinte:

- Criação dos dados, valores e regras relativos ao negócio do cliente no *Oracle Engagement Cloud*;
- Criação e gestão de campos e respetivas regras de negócio, visibilidade e obrigatoriedade no *OEC;*

- Integração de diversos *Web Services* e *APIs* e posterior validação e tratamento dos dados recebidos/enviados;
- Desenho e demonstração ao cliente das soluções desenvolvidas;
- Aplicação de diversas funcionalidades como *milestones*, canais de comunicação, filas de trabalho e atribuição automática, etc.;
- Realização de testes e correção de *bugs* (com e sem automação);

1.4 Enquadramento Institucional

Em suma, este capítulo pertente ilustrar, com base num artigo focado na qualidade do *CRM* e *CX* (*Customer Experience*) em diversas empresas no mercado, que a Accenture é uma empresa com bom historial e qualidade para desenvolver este tipo de projetos.

Figura 1."Magic Quadrant for CRM and Customer Experience Implementation Services, Worldwide" [7]

A imagem acima (Figura 1) é uma representação gráfica do chamado "Quadrante Mágico de Implementação de *CRM* e de Serviços de *User Experience*". O esquema foca apenas a implementação de sistemas *CRM* e de serviços de "*Customer Experience*" (*CX*), isto envolve, consultoria e produção e implementação de soluções. Essas soluções de *CRM* e *CX* requerem cada vez mais um amplo conjunto de disciplinas, sejam elas técnicas, de consultoria, de gestão ou negócio, e todas devem interatuar perfeitamente.

O quadro, produzido por analistas da *Gartner*, tem em conta diversas variáveis dentro das empresas, metodologias de trabalho, eficácia e qualidade das soluções, competitividade, e por fim, capacidade de capitalizar sobre uma visão. As variáveis anteriores estão refletidas no eixo vertical do referencial, para o eixo horizontal considera-se a capacidade das empresas em permanecerem atuais e progredirem tecnologicamente, ou seja, o entendimento que têm de como explorar as forças de mercado a favor do cliente.

Deste modo, os líderes neste "Quadrante Mágico" são aqueles que possuem um vasto leque de capacidades analíticas, técnicas e de negócio, de estratégia *CX* e *CRM* e todas as áreas tecnológicas, de consultoria e de gestão voltadas para os serviços de cliente. Naturalmente, as empresas no quadrante superior direito são aquelas que demonstram maior crescimento e lucros comparativamente à competição. Além disso, são também empresas que sobressaem em múltiplas regiões do globo com elevados níveis de satisfação dos clientes. [4]

1.5 Contribuições

Enquanto entidade individual na equipa do projeto, a minha contribuição foi significativa. Com isto pretendo dizer que, para a globalidade do projeto implementado, o meu trabalho em específico é mensurável e teve impacto. Isto pode ser verificado em grande parte pelo volume de trabalho realizado por mim. Não é fácil enumerar, na totalidade, todas as contribuições feitas para o projeto e para a equipa, deste modo, mencionarei apenas as principais. Mais à frente, no terceiro capítulo, teremos então oportunidade de analisar mais aprofundadamente as especificidades do trabalho realizado por mim.

Sabendo que este projeto passa muito pela configuração de um sistema base, muito do meu trabalho prendeu-se com essa mesma configuração. Esta configuração pode ser feita através de programação, com a escrita de *scritps* na linguagem *Groovy*, como por exemplo. em funções de validação do NIF, funções de gestão das categorias, funções para chamadas e integração de W*eb Services*, etc. A criação de campos e formulários é também uma configuração base para o sistema pois é sobre eles que todas as operações assentam. Ao nível da criação de campos, criei à volta de 350 distintos. A importação de

listas de valores para a aplicação é também algo fundamental, em especial as listas das categorias, trabalho também desenvolvido por mim. Outro aspeto essencial da aplicação são os perfis de segurança, novamente, também configurados em grande extensão por mim. Todos os trabalhos realizados e mencionados até agora estão centrados na aplicação utilizada, o *Oracle Engagement Cloud.* Mas a contribuição mais importante foi uma página *web Java* desenvolvida sobre a aplicação, apesar de não constar no planeamento inicial. Para já, basta dizer que esta página alberga cerca de 300 dos 350 campos criados na aplicação e aplica sobre eles as complexas regras de obrigatoriedade e visibilidade do negócio. Além disso, esta página está embebida diretamente na aplicação e comunica com ela por pedidos *REST*. Mais uma vez, não estou a nomear nem a elaborar exaustivamente todo o trabalho desenvolvido, apenas os aspetos principais. Fazendo um ponto de comparação com o capítulo 1.2 dos objetivos, o trabalho conseguido:

- Importação dos dados e valores das categorias e dos produtos;
- Criação de funções de negócio para gestão dos objetos e valores importados;
- Criação e gestão de campos e respetivas regras de negócio, visibilidade e obrigatoriedade no *OEC;*
- Desenvolvimento de uma aplicação *web* em *Java* para o preenchimento e validação de campos, com integração com serviços e *APIs* externas, com um *autocomplete* de moradas, etc.;
- Integração de diversos *Web Services* e *APIs* e posterior validação e tratamento dos dados recebidos/enviados no *OEC*;
- Participação no desenho e apresentação ao cliente do trabalho desenvolvido com particular enfase sobre um dos *sub-sprints* onde fui o principal responsável;
- Aplicação de diversas funcionalidades no *OEC* como *milestones*, canais de comunicação, filas de trabalho e atribuição automática, etc.;
- Desenho de uma possível solução para automação de testes (não implementada);

1.6 Estrutura do documento

Além do capítulo introdutório atual, este documento irá ainda integrar os seguintes capítulos:

Capítulo 2: Objetivos e Metodologias

Capítulo 3: Trabalho Realizado

Capítulo 4: Conclusões

No segundo capítulo será abordada a implementação da componente Sales do *CRM* também no *OEC*. No terceiro capítulo, o capítulo principal, é explicado todo o trabalho realizado até ao momento. Sendo que, é feita uma separação entre trabalho realizado diretamente na aplicação *Oracle Engagement Cloud* e fora desta. Ainda no capítulo do trabalho realizado, está explicada a preparação dos testes automáticos ao *OEC*, bem como, ainda que muito brevemente, o trabalho de análise de requisitos e produção de documentação. E por fim, o quarto e último capítulo, as conclusões, onde é feito um sumário de todo o documento com algumas opiniões particulares relativamente a todo o trabalho.

Capítulo 2

Objetivos e Metodologias

2.1 Contexto Subjacente

No momento em que integrei o projeto, estava já em vigor, isto é, em fase de produção, a componente de *Sales* do *CRM*. Esta componente, como referido anteriormente, surge da necessidade do cliente em gerir a sua carteira de utilizadores, catálogo de produtos e processo de vendas. Isto permite aos funcionários da área comercial gerir oportunidades de negócio e acompanhar o seu ciclo de vida de forma integrada com outras plataformas externas. Ou seja, uma aplicação que melhora a gestão de clientes e dinamiza as vendas da empresa ao mesmo tempo que integra e unifica diversos sistemas externos.

Figura 2. Menu principal do Oracle Engagement Cloud, aberto na secção Sales

Como é possível observar na figura acima (Figura 2), o separador principal "*Sales*" está selecionado, revelando todas as subcategorias da componente. Em especial, as "*Leads*" que são os possíveis contactos ainda por registar no sistema, as "*Opportunities*" que são os contactos já presentes no sistema e que representam

oportunidades de vendas, sendo que as "Leads" precedem as "*Opportunities*" [2]. As "*Accounts*", onde se podem criar e gerir as diversas contas presentes no sistema. O "*Contacts*", semelhante às "*Accounts*", permite criar e gerir contactos. De salientar que as contas e contactos são a base do sistema, pois a empresa depende em grande parte da sua carteira de contactos e da sua eficiente gestão.

A implementação da componente *Sales* está intimamente relacionada com a componente em implementação no presente relatório, o *Service*. Como se pode ver ainda na imagem anterior, o menu *Service* está imediatamente à direita da secção Sales, pois a mesma aplicação integra as duas componentes. Mais do que isso, os objetos (estruturas que serão explicadas em detalhe mais à frente neste relatório) do *Sales* são utilizados e partilhados com o *Service*. Isto é, e a título de exemplo, as contas e os contactos, responsabilidade da implementação de *Sales*, são os mesmos que são utilizados no *Service* e nos seus objetos. Todos os objetos estão, portanto, disponíveis e partilhados entre as duas componentes.

2.2 Metodologias

O projeto é desenvolvido segundo uma metodologia *Agile*, mais especificamente, *Scrum*. Encontra-se por isso, divido em sprints, isto é, delimitações temporais para a entrega de determinados conjuntos de funcionalidades. Neste caso em específico existem três *sprints* com as seguintes especificações:

- **Sprint 1**: Gestão de Solicitações e Interações do Cliente
- **Sprint 2**: Gestão de Pedidos de Recolha/Segundas Entregas/Entregas no Próprio Dia e Incidências Operacionais
- **Sprint 3**: Visão 360º do Cliente/ *Knowledge Base* e *Reporting*

Estão também presentes os três papeis típicos da metodologia *Scrum*, o *Product Owner*, que está representado por duas pessoas pertencentes ao cliente. Estas pessoas são então as responsáveis por transmitir a visão do negócio que se pretende implementar. O *Scrum Master*, que está presente na forma da *Manager* do projeto, e que é a pessoa responsável por facilitar a chegada da informação do *Product Owner* até à equipa. É a pessoa que deve ser a ponte entre a componente do negócio e a componente técnica (equipas de implementação) do projeto. Por fim, a própria equipa, que é quem realmente implementa a visão do *Product Owner*. [8]

Agora, no contexto mais técnico, e ao nível da aplicação em si. O projeto, como já referido, é uma solução *CRM* para uma empresa do ramo dos serviços postais. Aquando da integração na equipa do projeto tinha já sido implementada a componente

de *Sales* do *CRM*, sendo então a componente de *Service* a nova etapa a desenvolver. A solução divide-se em três camadas principais, *Dev* (Desenvolvimento), Qualidade e Produção. *Dev* é a camada onde a equipa e eu operamos e será por isso a camada principal daqui em diante. Qualidade é onde se efetuam os testes finais, por parte do cliente e em colaboração com alguns membros do projeto. Por último, Produção, que representa a fase de implementação do projeto, já no cliente e em utilização prática. Em *Dev* importa ainda referir que existe uma metodologia de trabalho faseada, isto é, uma fase inicial de teste e preparação da solução, e a fase de publicação real para o ambiente de *Dev*. Esta estrutura é utilizada para os trabalhos desenvolvidos diretamente no *Engagement Cloud* através de um sistema de "*Sandboxes*". As "*Sandboxes*" no *OEC* têm um funcionamento semelhante ao de um repositório de controlo de versões, tirando o facto de não guardar versões diretamente. Ou seja, existe uma "*Mainline*" onde está a versão mais atual do ambiente, e existem as "*Sandboxes*" que são ramificações da "*Mainline*" (cópias) para cada um de nós trabalhar e realizar as alterações necessárias. Uma vez publicada (ação semelhante a um "*Commit*") a "*Sandbox*" passa a ser a versão global do ambiente da aplicação, e todas as outras "*Sandboxes*" criadas antes da publicação ficam desatualizadas. Estando desatualizadas não devem ser publicadas para não eliminar as alterações feitas na última publicação, sendo por isso necessário criar uma nova "*Sandbox*" caso haja interesse em publicar alguma alteração posterior.

2.3 Trabalho Planeado

O trabalho planeado para este projeto está, como já referido, muito relacionado com a configuração de uma aplicação *CRM* na *cloud.* Para o planeamento deste projeto o essencial é integrar os diversos sistemas e aplicações independentes que o cliente possui atualmente e unificar todas elas num único centro de operações. Isto é, tornar o *OEC* a ferramenta de trabalho comum a todos os setores da empresa cliente, independentemente das diferentes aplicações e métodos de trabalho que utilizem. Para atingir esta meta, um dos planos de trabalho adotados consistiu em realizar sessões de prototipagem junto do cliente para apresentar o trabalho já desenvolvido. Por outras palavras, o trabalho foi realizado em pequenos "*sub-sprints"* (inseridos dentro dos *sprints* já mencionados) que culminavam com a apresentação ao cliente do progresso. Nestas sessões eram apresentados alguns cenários de utilização das funcionalidades introduzidas, eram discutidas as divergências face à visão do cliente e apresentadas possíveis soluções. Este planeamento do trabalho enquadra-se com a metodologia *Agile* em vigor no projeto.

Relativamente ao trabalho em si, o plano é o que se pode observar na Figura 3, com especial ênfase nos dois indicados com as setas vermelhas, os trabalhos dentro do *Oracle Engagement Cloud* (*OEC*) – *Service*, e os testes de integração e aceitação. O

trabalho no *OEC*, inclui não só as configurações dentro do sistema, como também todas aquelas feitas fora dele para o complementar. Isto é, uma vez que o *OEC* nem sempre se mostrou capaz de resolver certos problemas ou requisitos maios complexos do cliente, foi necessário optar por opções externas "à medida". Ou seja, inicialmente estava previsto aplicar apenas o *OEC*, no entanto tornou-se necessário recorrer a aplicações *Web* externas, mais concretamente a página desenvolvida para gerir os campos e as suas regras e validações. Relativamente aos testes de integração e de aceitação, estes passavam por realizar validações ao sistema tanto manual como automaticamente. Infelizmente, e apesar de ter existido uma tentativa de implementar os testes automáticos no sistema, algo que será falado em detalhe mais à frente, não o foi possível fazer pois isto ocuparia tempo e recursos que não estavam ao dispor no momento.

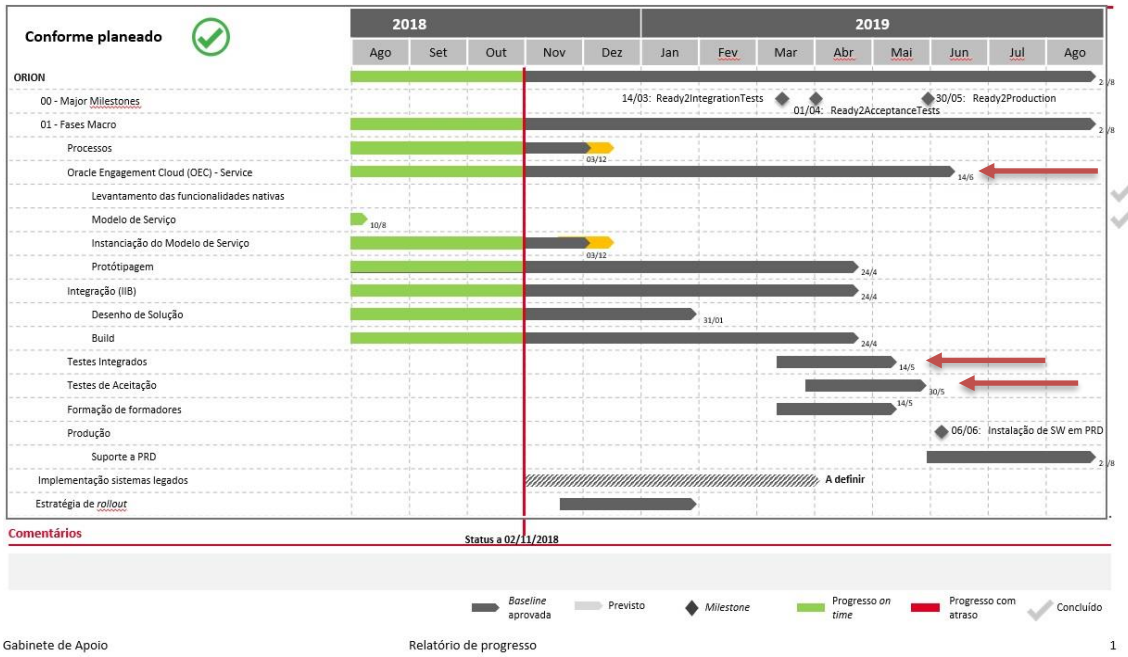

Figura 3. Planeamento do Projeto para o período do meu estágio

O projeto pressupunha alguns marcos chave no seu desenvolvimento, o primeiro era a entrada do trabalho na fase de testes de integração em março de 2019. Esta nova fase do projeto e respetivo prazo eram específicos para o primeiro *sprint*, e dependia da sua conclusão. E para o *sprint* ser concluído a equipa teria que ter implementado, no ambiente de desenvolvimento (*dev*) todas as configurações gerais, mais as configurações específicas à gestão de solicitações e interações com o cliente. Por outras palavras, configurar o sistema para receber e tratar os pedidos dos clientes, bem como as suas possíveis reclamações. Isto passa, de forma genérica, pela configuração de emails para canais de entrada de solicitações, na construção de fluxos automáticos de resposta e

criação de pedidos, na preparação das categorias e a sua gestão, preparação dos campos associados a cada tipo de pedido do cliente e respetivas regras e validações, preparação dos perfis com restrições de segurança para cada tipo de utilizador, entre outros. Todo este trabalho e subsequente entrada em testes de integração culminou em maio de 2019. O ligeiro atraso deveu-se maioritariamente ao atraso na entrega de alguns dados essenciais para o primeiro *sprint*, por parte do cliente. Por exemplo as listas de valores para categorias, produtos, canais de entrada, etc.

Com a entrada numa fase de testes surge a necessidade de acompanhar os *testers* e realizar as correções necessárias.

O marco seguinte é a passagem do trabalho realizado para o ambiente de qualidade, no fundo, é "passar a limpo" tudo o que foi produzido no ambiente de *dev*. Dado isto, dá-se a entrada em testes de aceitação. A grande diferença entre estes testes e os anteriores de integração, é que estes são realizados semi-autonomamente pelo cliente, ao contrário dos primeiros que são realizados por membros da equipa funcional do projeto. A outra diferença é que estes acontecem a uma escala muito maior, isto é, com mais utilizadores a testar, e num ambiente de utilização quase real. Dado o atraso inicial, é natural que todos os prazos doravante não sejam cumpridos, sendo que estes marcos foram atingidos em finais de junho de 2019, com a passagem das configurações para o ambiente de qualidade. Em paralelo, começa também a decorrer o segundo *sprint*, que sendo mais pequeno, tal como o terceiro, não ocupam tanto tempo nem recursos da equipa.

Finalmente, após todas as afinações decorrentes dos testes de aceitação o produto é passado para o ambiente de produção, ou se preferirmos, para o dia-a-dia dos seus utilizadores. Esta passagem está planeada para ser automática, isto é, uma cópia direta do ambiente de qualidade, o que reduz em muito o tempo e esforço necessários para essa transição. Esta fase, deverá acontecer no início de setembro de 2019, altura essa que já não é abrangida pelo presente relatório.

2.4 Ferramentas Utilizadas

No contexto deste trabalho foram utilizadas diversas ferramentas e tecnologias, quer sejam estas linguagens de programação, *frameworks* ou algum tipo de *software*. Este capítulo serve por isso como guia às diversas ferramentas e tecnologias que serão abordadas ao longo deste relatório. No entanto, este capítulo não fará referência á ferramenta principal, o *OEC*.

2.4.1 Linguagens

- ***Groovy***: É uma linguagem de programação orientada a plataformas *Java*. É a linguagem utilizada no *Oracle Engagement Cloud* para redigir *scripts* e funções e distingue-se pela sua simplicidade e rapidez de implementação.
- ***JSON***: *JavaScript Object Notation*, é uma linguagem utilizada para descrever objetos e dados. Utilizada por exemplo para a transmissão de dados estruturados entre serviços e aplicações.
- ***Java***: É uma linguagem de programação que dispensa apresentações. É boa para desenvolver *software* como aplicações *Web* e tem a vantagem de integrar facilmente com diversas *frameworks* e bibliotecas de desenvolvimento.
- ***JavaScript***: É uma linguagem de programação de alto nível geralmente associada à programação *Web*. Neste contexto, a linguagem é executada ao nível do *browser*.
- ***JQuery***: É uma ramificação do *JavaScript* que facilita a ligação com elementos do *HTML* e a gestão de eventos nas páginas.
- ***HTML***: É uma linguagem de anotação de documentos extremamente comum no desenvolvimento e visualização de páginas *Web*.
- ***CSS***: É uma linguagem associada ao *HTML* e é responsável por definir o estilo e apresentação dos documentos.
- ***SQL***: É uma linguagem utilizada para comunicar com bases de dados relacionais. Foi, neste caso, a linguagem utilizada para fazer pesquisas dentro do *Oracle BI*.

2.4.2 Frameworks e Tecnologias

- ***Maven***: É uma ferramenta de automação e aceleração do desenvolvimento de projetos em *Java*. Tem a grande vantagem de integrar e gerir, rapidamente, quaisquer outras *frameworks* e ferramentas de desenvolvimento, que tomam o nome de dependências. No fundo é uma ferramenta de gestão dos recursos de um projeto.

- ***Spring***: É uma *framework* que toma o controlo dos objetos da aplicação utilizando apenas anotações. É uma forma de acelerar o desenvolvimento e arquitetura de uma aplicação, em particular com as extensões para aplicações *Web*.
- ***Thymeleaf***: É uma ferramenta que integra diretamente com o *Spring* e serve como motor entre o *back end* e *front end*. É utilizado nas aplicações *web* para gerir a informação nos documentos *HTML*.
- ***Bootstrap***: É uma *framework* de *CSS* aberta, e serve para acelerar o desenvolvimento do *front end* de uma aplicação. Permite criar documentos *HTML* responsivos, limpos e uniformes.
- ***Cucumber/Gherkin***: O *Cucumber* é uma ferramenta de desenvolvimento orientada aos comportamentos. O seu interpretador reconhece a sua linguagem de anotações própria, o *Gherkin*. No fundo o que esta ferramenta permite é identificar e decompor em passos lógicos um conjunto de comportamentos.
- ***Selenium***: É uma *framework* de testes para aplicações *Web*. Esta ferramenta permite executar um conjunto de passos descritos diretamente na aplicação e registar e produzir relatórios sobre os mesmos.
- ***JUnit***: É uma *framework* de testes para aplicações *Java* que permite executar código para testes funcionais. Apesar de ter muito mais funcionalidades, neste projeto apenas foi utilizado para executar a aplicação como uma aplicação de testes.

As *frameworks* acima listadas podem ser separadas em duas componentes, a componente das aplicações *web* e a componente das aplicações para automação de testes. Para a primeira, temos as ferramentas *Maven*, *Spring*, *Thymeleaf* e *Bootstrap*, para o segundo caso temos o *Maven*, *Cucumber*/*Gherkin*, *Selenium* e *JUnit*.

2.4.3 Software

- ***Postman***: É um *software* que permite, entre outras coisas, executar chamadas a *Web Services* tanto *REST* como *SOAP*. É útil para executar chamadas às *APIs* expostas nos endereços *URL* e comunicar com outros sistemas e aplicações sem ter que traduzir de imediato as chamadas para código.

- ***Eclipse***: É um *IDE (Integrated Development Environment)* de programação, muito comum no desenvolvimento de aplicações *Java*.
- ***Oracle BI***: É uma ferramenta de *Business Intelligence* para análise e apresentação de dados e informação dos sistemas de negócio. Neste projeto em particular os dados são os produzidos e armazenados no *OEC*.
- ***Microsoft Office***: Em particular o *Excel* que teve especial impacto na análise de dados e requisitos e na preparação de ficheiros de importação de valores.
- ***TortoiseSVN***: É um sistema de controlo de versões para código e projetos e funciona sobre o *Windows*.
- ***Microsoft Teams***: É uma ferramenta de comunicação e partilha de informação entre uma ou várias equipas. Oferece ainda algumas componentes de gestão.

As opções tomadas sobre as tecnologias a utilizar foram em grande parte condicionadas tanto pela tecnologia principal a ser utilizada, o *OEC*, bem como pelas regras e estrutura definidas para as aplicações customizadas. Por exemplo, o *OEC*, impõe que os seus serviços sejam *REST* e não *SOAP*, ou que a sua linguagem seja o *Groovy*, deste modo, não é possível fugir a estas ferramentas. No lado das aplicações à medida, todas elas tiveram que ser desenvolvidas segundo algumas regras estruturais e arquiteturais previamente definidas, e como tal a opção sobre as ferramentas utilizadas também não decaiu sobre mim. No entanto, podemos falar em algumas alternativas às tecnologias utilizadas. O *Maven*, por exemplo, tem um grande rival, o *Gradle*, ambas as ferramentas têm propósitos semelhantes, e apesar do *Gradle* ser mais simples e poderoso, tudo isso não consegue abafar a experiência e o nome detidos pelo *Maven*. No final, o *Maven* é a ferramenta com mais suporte, mais conhecida e com mais provas dadas de funcionamento [9]. Outro exemplo é a *framework Spring* e as suas alternativas *JSF (Java Server Faces)* ou o *Hibernate*. Apesar de semelhantes, o *Spring* tem uma comunidade e suporte maiores, boa limpeza e organização do código, é adequado para desenvolvimento ágil e tem um nome estabelecido no mercado. Tal como para ambos os casos exemplo, o mesmo acontece para várias outras ferramentas apresentadas.

Capítulo 3

Trabalho Realizado

Este capítulo aborda em detalhe o trabalho realizado por mim no decorrer deste projeto. De relembrar, no entanto, que nem sempre serão aprofundados assuntos menos relevantes, como por exemplo, a criação de campos numa entidade, é um processo repetitivo e de baixa complexidade, e como tal será mencionado por alto. O mesmo sucederá para todas as contribuições realizadas deste género. Outra ressalva vai para o facto de nem sempre ser possível aprofundar um tema por se tratar de informação mais particular sobre o cliente. Este tipo de situação poderá ocorrer principalmente ao nível das chamadas aos *Web Services*, uma vez que podem identificar negócio direto do cliente.

3.1 Trabalho no *Engagement Cloud* (*Oracle*)

Como já falado anteriormente, o trabalho direto no *OEC* (*Oracle Engagement Cloud*), isto é, o trabalho que é realizado dentro da aplicação, passa, em traços muito gerais, pela customização da aplicação base. Isto envolve criação de campos, regras de negócio e *triggers*, gestão de chamadas externas (por *Web Services*), etc.

3.1.1 Criação de Campos

A aplicação possui objetos, que podem ser entendidos como as diversas classes do sistema, como por exemplo o objeto "Contacto", "Conta", "Pedido de Serviço", etc. Todos os objetos nomeados são "*Standard*" ou "*Out-of-the-Box*", isto é, são objetos que vêm predefinidos na aplicação e que se esperam ser suficientes para a generalidade das implementações *CRM*. Para além dos objetos "*Standard*", temos ainda os objetos "*Custom*" que são objetos criados por quem desenvolve para cumprir requisitos inatingíveis apenas com os objetos base.

Dentro de cada objeto, existem campos ("*fields*"), que, como os atributos das classes podem ser de vários tipos, como se pode ver na Figura 4, abaixo.

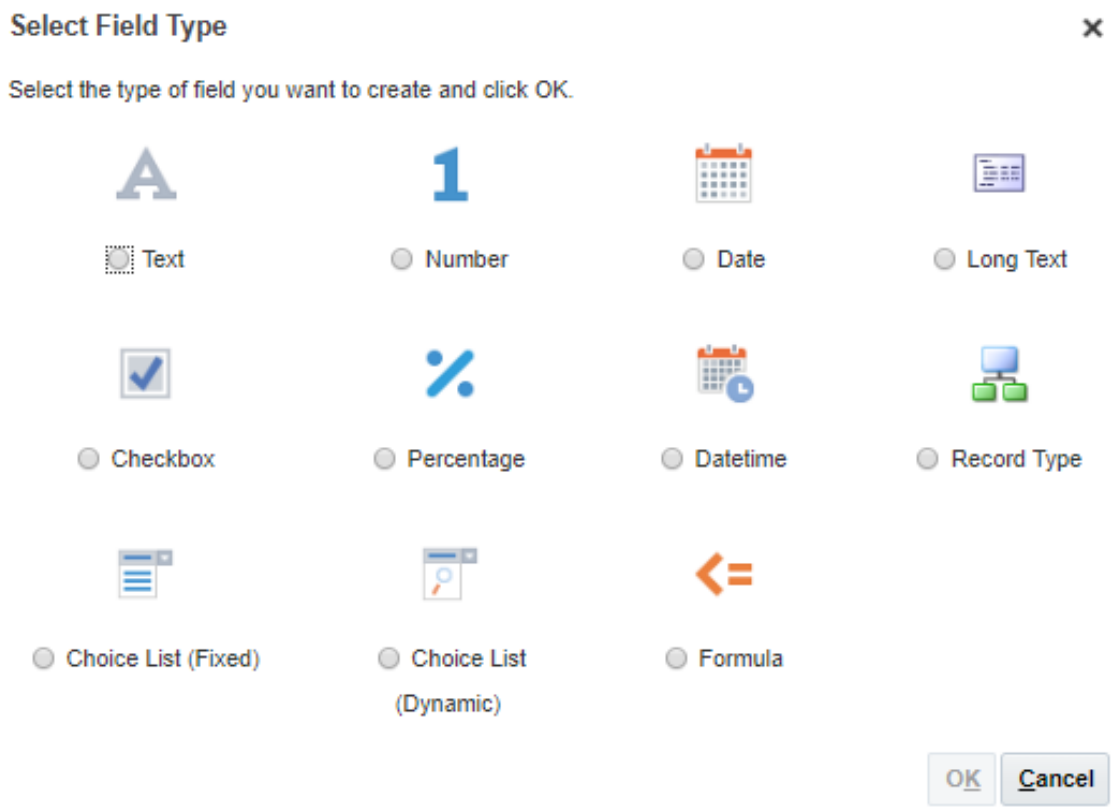

Figura 4. Tipologia dos Campos no OEC no menu de criação de um campo

Os campos, tal como os objetos, também se dividem em dois tipos, "*Standard*" e "*Custom*". Novamente, os "*Standard*" são os campos que figuram num objeto por *default* da aplicação, e os "*Custom*" aqueles que o programador cria e customiza (Figura 5).

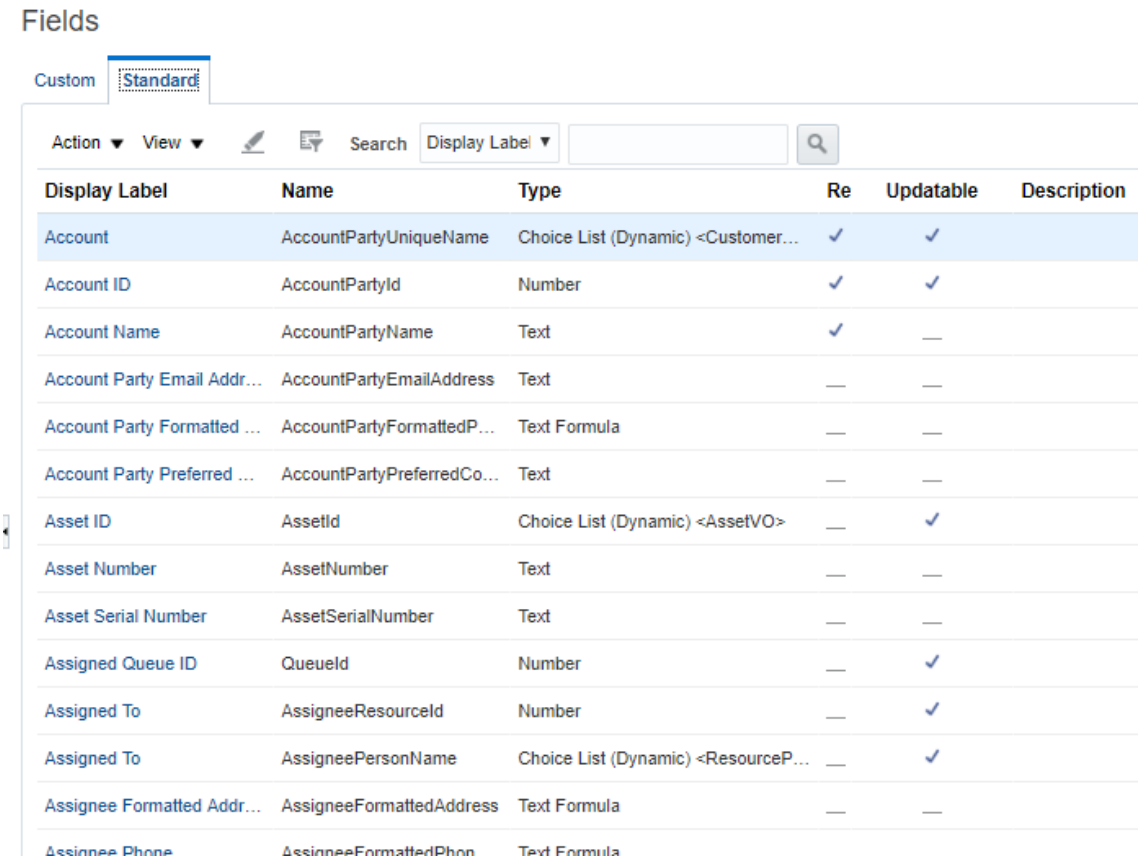

Figura 5. Lista de Campos Standard do Objeto Service Request

3.1.2 Criação de *Triggers* e Funções

Os *triggers* no *Oracle Engagment Cloud*, tal como na programação e nas bases de dados, são ações ou funções que "disparam" quando um determinado evento ocorre. Os "*Triggers*" podem ser "*Field Triggers*" ou "*Object Triggers*". Os "*Field Triggers*" observam diretamente um campo específico do objeto. Ou seja, dentro de um objeto podemos definir "*Triggers*", e podemos especificar se o mesmo dispara apenas sobre a alteração de um determinado campo, ou se para todo o objeto. Nesse caso, utiliza-se um "*Object Trigger*", que por sua vez pode ter diversos tipos. Os tipos do "*Object Trigger*" estão associados às fases de existência dos dados de uma instância do objeto. "*Before Create*", "*Before Insert*", "*Before Update*", "*After Create*", etc. (Figura 6). A ideia é podermos disparar um *script* em qualquer estádio da vida dos dados. Idealmente existiria apenas um "*Trigger*" para cada estado que englobaria todas as operações para esse mesmo estado. Se tal não for possível e acabarmos com mais do que um "*Trigger*" por estado, então a ordem pela qual eles irão disparar depende da ordem com que aparecem na lista de "*Triggers*", que é descendente (de cima para baixo).

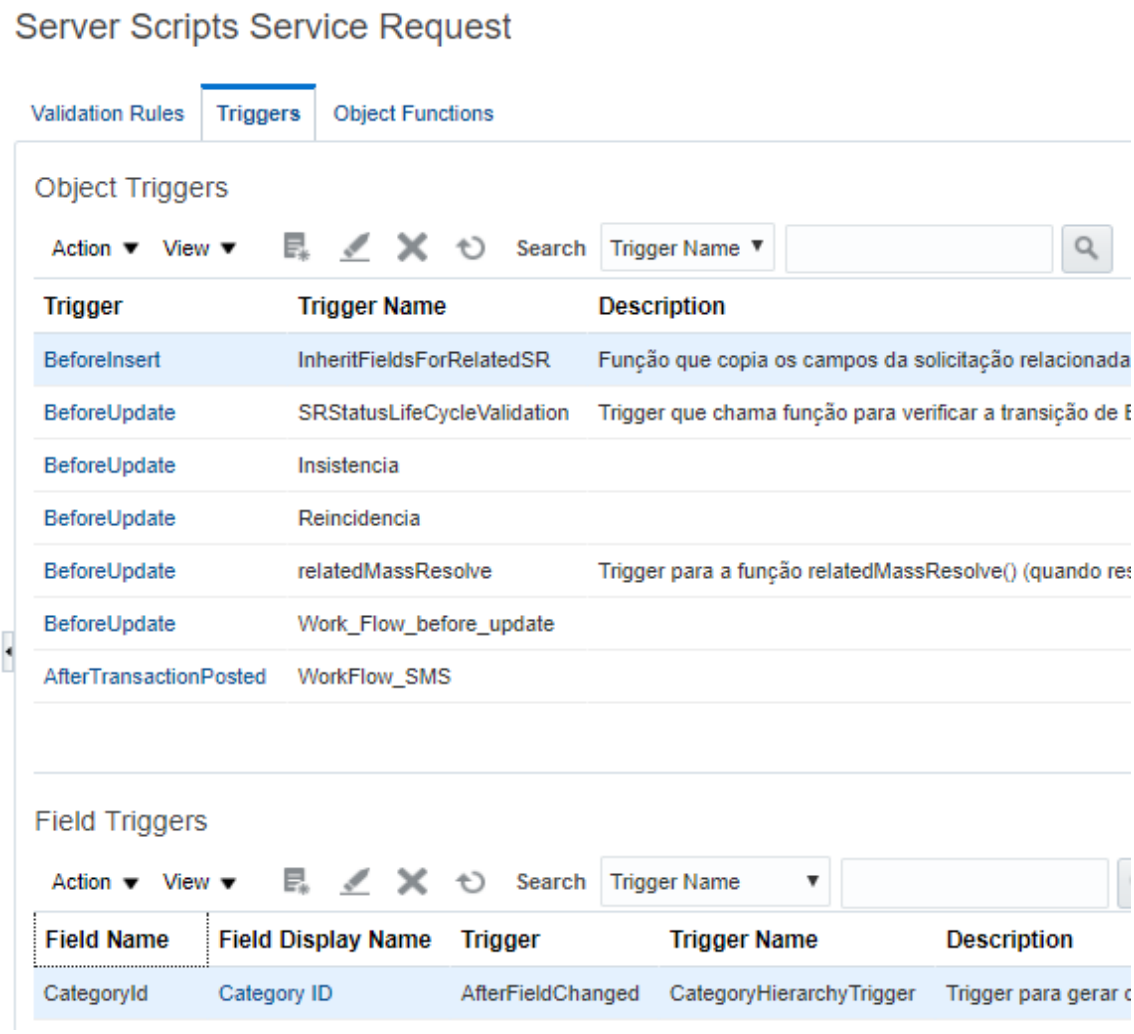

Figura 6. Página de configuração dos triggers do Objecto Service Request, Opções Criar/Editar/Apagar

No entanto, esta forma de aplicar os *triggers* não é a mais correta nem elegante, uma vez que o evento que dispara o *trigger* irá ser verificado diversas vezes. Assim, a melhor abordagem é a de ter apenas um *trigger* por cada um dos tipos existentes, e apenas lá dentro chamar as funções pretendidas. Esta foi uma das "limpezas" ao código da aplicação que foram efetuadas por mim.

As "Regras de Validação" e as "*Object Functions*" são semelhantes aos "*Triggers*", no sentido em que todos eles são pedaços de código em "*Groovy*". No entanto, as "*Object Functions*" são funções globais ao objeto e que servem essencialmente para ser invocadas nos "*Triggers*" e/ou "Regras de Validação" do objeto em que foram definidas, não confundir com as funções globais. As funções globais são exatamente iguais às "*Object Functions*" tirando que não estão confinadas num objeto, estando assim disponíveis para todos os objetos. Relativamente às "Regras de Validação", estas são como "*Triggers*" estando também divididas em "*Object Rules*" e "*Field Rules*", sendo que o seu propósito é validar os dados inseridos quer num campo específico quer

no objeto como um todo. Devemos optar pela sua utilização face aos "*Triggers*" sempre que o objetivo for somente a validação de dados, como por exemplo o NIF, ou um email.

3.1.3 Linguagem *Groovy* e *Scripts*

Já falámos das funções e dos "*triggers*", mas ainda não abordámos a linguagem utilizada para os definir. O *Groovy* é uma linguagem de programação orientada a objetos desenvolvida para ser uma alternativa ao *Java*. Enquanto linguagem apresenta algumas semelhanças com *Python*, Ruby e naturalmente, o *Java*. O *Groovy* comum, integra-se, de forma transparente, com outros códigos e bibliotecas do *Java*. Apesar disto, o *Groovy* utilizado no *Oracle Engagement Cloud* está no seu estado puro, isto é, não permite importar ou aceder a quaisquer bibliotecas externas, uma opção que confere robustez e segurança à aplicação, tirando-lhe, no entanto, elasticidade.

```
if(CategoryId!=null && CategoryId!=''){

 def conn = adf.webServices.CategoryHierarchyWebservice
 def data = null
 try{
    data = conn.GET(CategoryId.toString())

    if(data['LeafNodeFlag'] == false){
      adf.userSession.userData.put('isLeafNodeCategory', false)
    }else{
      adf.userSession.userData.put('isLeafNodeCategory', true)
    }

    def hierarchy = null, level1 = null, level2 = null,
      level3 = null, level4 = null, level5 = null;
    hierarchy = data['Hierarchy']
    def catSplit = data['Hierarchy'].split(' > ')
    def size = data['Depth']

    switch(size) {
        case "0":
        level1 = data['CategoryName']
        hierarchy = level1
        break

        case "1":
        level2 = catSplit[1]
        level1 = catSplit[0]
        break

        case "2":
        level3 = catSplit[2]
        level2 = catSplit[1]
        level1 = catSplit[0]
        break
```

Figura 7. Exemplo de Script Groovy, desenvolvido para o tratamento das categorias (Parte 1)

```
            case "3":
            level4 = catSplit[3]
            level3 = catSplit[2]
            level2 = catSplit[1]
            level1 = catSplit[0]
            break

            case "4":
            level5 = catSplit[4]
            level4 = catSplit[3]
            level3 = catSplit[2]
            level2 = catSplit[1]
            level1 = catSplit[0]
            break

            default:
            level1 = null
            level2 = null
            level3 = null
            level4 = null
            level5 = null
            break
        }

        setAttribute('CategoryHierarchy_c', hierarchy)
        setAttribute('CategoryLevel1_c', level1)
        setAttribute('CategoryLevel2_c', level2)
        setAttribute('CategoryLevel3_c', level3)
        setAttribute('CategoryLevel4_c', level4)
        setAttribute('CategoryLevel5_c', level5)

      }catch (e) {
        throw new oracle.jbo.ValidationException(
          'Error in Category Hierarchy Trigger  '+
          + e.toString())
      }
}
```

Figura 8. Exemplo de Script Groovy, desenvolvido para o tratamento das categorias (Parte 2)

O código anterior, figuras 7 e 8, é um exemplo de uma chamada a um *Web Service* e posterior utilização da informação para preencher campos do objeto "*Service Request*". Neste exemplo de código desenvolvido, é feito um pedido *GET* através do *Web Service* "*CategoryHierarchyWebservice*" também por mim definido no âmbito do projeto. Em baixo é possível ver, através da ferramenta "*Postman*" qual a estrutura geral de uma "Categoria". Esta ferramenta é útil para testar pedidos *REST* e *SOAP* a *APIs* e serviços externos antes de serem incorporados no *Engagement Cloud* (Figura 9). Neste caso, podemos ver os nomes dos campos que serão recebidos, no formato de um mapa de valores, com a estrutura do *JSON* apresentado. Assim, tendo o nome dos campos (chaves do mapa) e o mapa, podemos facilmente obter os valores que pretendemos.

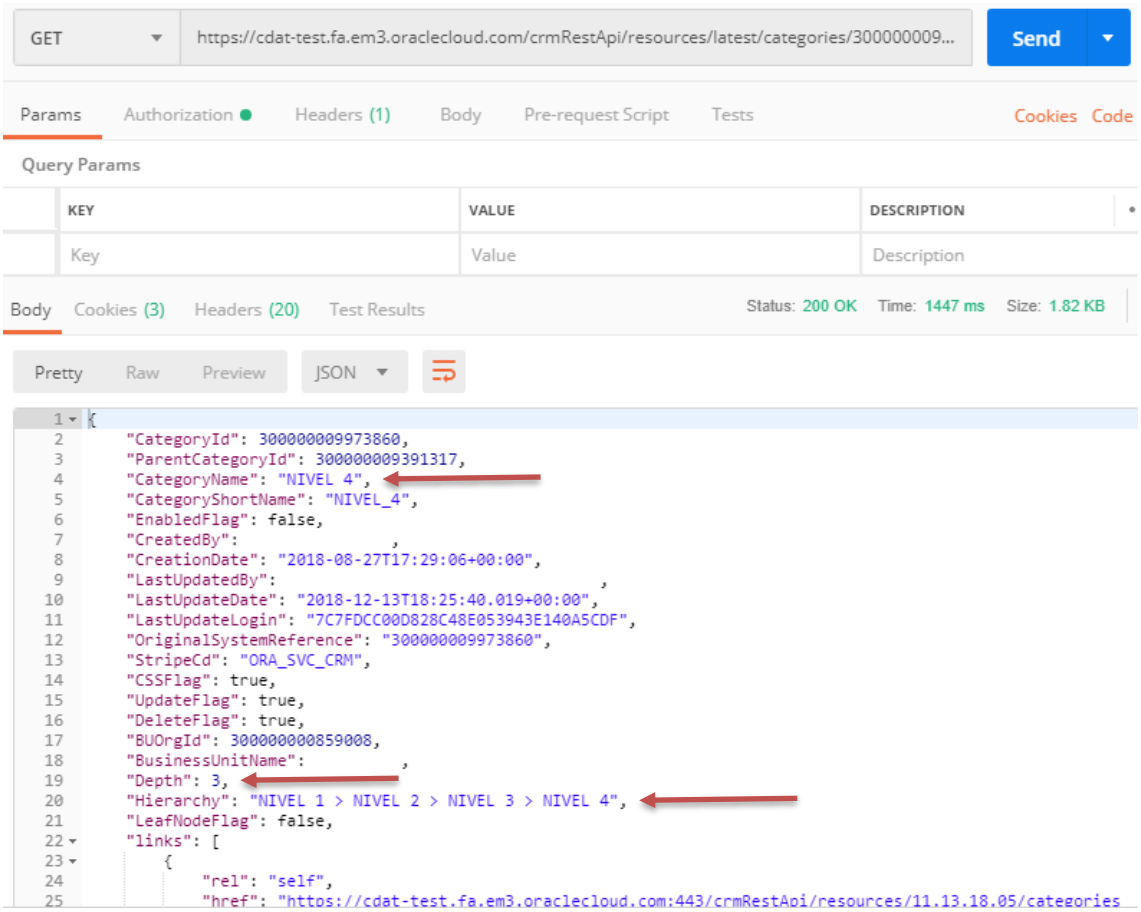

Figura 9. Ecrã da aplicação Postman, com o resultado do pedido GET para a categoria "NIVEL 4"

3.1.4 *Web Services* e *APIs*

Na secção anterior deu-se um exemplo de um script *Groovy* para obter a hierarquia de categorias acima de uma dada categoria. Esse script está incorporado num "*Field Trigger*", que, como já vimos, dispara aquando da alteração de um campo. Neste caso, o campo que estamos a observar é o campo "*CategoryId*", que é atualizado cada vez que seja selecionada uma categoria diferente na *dropdown* da Categoria (como se pode observar em baixo na Figura 10).

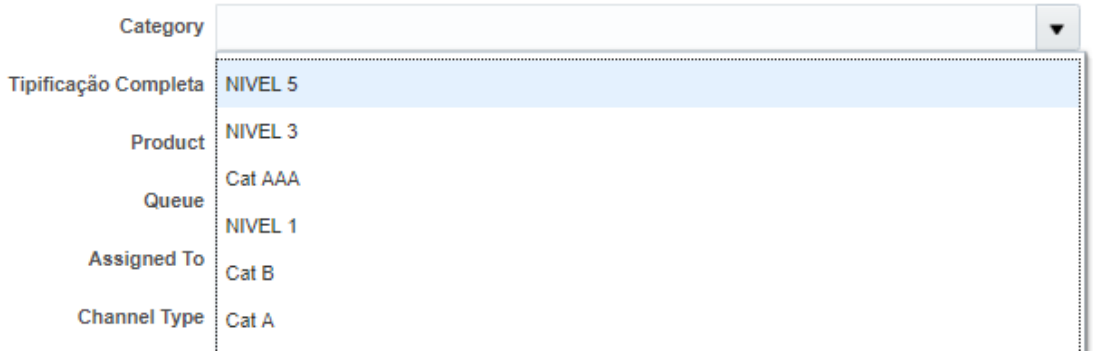

Figura 10. Dropdown do campo categoria no Service Request

Todos estes conceitos já foram abordados anteriormente, passemos por isso ao conceito dos *Web Services* no contexto do *Oracle Engagemente Cloud*. No código apresentado é feita uma chamada a um *Web Service* com o nome de *CategoryHierarchyWebservice*, mais concretamente um pedido *GET*. Essa ligação a esse *Web Services* foi implementada dentro do *OEC* na secção de "*Web Services*", como podemos ver na figura seguinte (Figura 11).

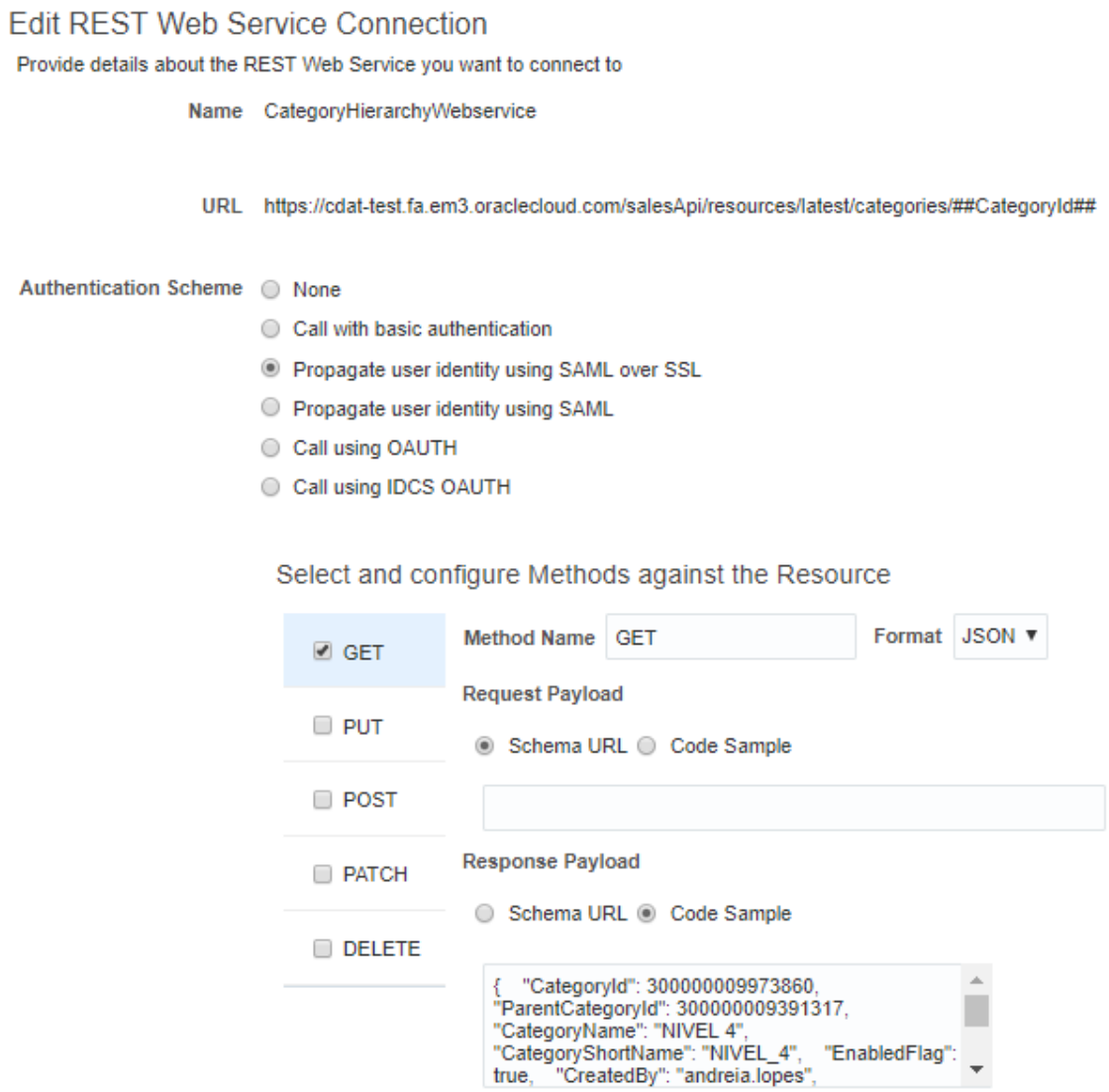

Figura 11. Ecrã de criação e edição de uma ligação com um Web Service dentro da aplicação

Na imagem podemos ver o menu de definição de uma ligação a um *Web Service* em *Oracle Engagement Cloud*. Atribui-se um nome e um *URL*, com a particularidade de que podemos enviar variáveis pelo *URL*, como os cabeçalhos de um pedido *REST*. Apenas que neste caso, a passagem de variáveis é feita preparando o *URL* do *Web*

Service para receber uma determinada variável, utilizando *"##NOME DO CAMPO##"* dentro do *URL*. De seguida define-se o esquema de autenticação, sendo que o mais comum é o esquema de propagação de identidade. Seleciona-se o método que pretendemos, neste contexto o *GET*, o formato da resposta, neste caso *JSON*, e definimos uma resposta tipo. É para a resposta tipo que o software "*Postman*" volta a ser útil, pois apenas precisamos de copiar a resposta JSON, aquela que vemos na figura 5., e colocá-la no *Response Payload* do *Web Service* que estamos a criar, dando assim um modelo tipo da resposta que estamos á espera de receber.

Agora que sabemos alguns dos conceitos do *OEC* e como funcionam podemos olhar para o exemplo na sua totalidade. Quando o campo "*Category*" é alterado o "*Field Trigger*" com o script *Groovy* exibido anteriormente dispara. Nesse script é feita uma chamada ao *Web Service* que definimos, passando-lhe a variável "*CategoryId*" que o URL do *Web Service* está à espera de receber. Assim, com o *URL* com o "*CategoryId*" torna-se possível fazer o pedido *GET* e obter o *JSON* que define aquela categoria. Tendo o *JSON*, que o *OEC* interpreta como um mapa, podemos obter o valor da hierarquia para essa categoria especifica. Em seguida, os valores são separados e utilizados para popular os campos necessários do objeto "*Service Request*".

Este caso apresentado foi desenvolvido como uma tarefa única, no entanto é um exemplo bastante completo das principais áreas do *Oracle Engagement Cloud* e permite ter um *overview* dos vários tipos de trabalhos realizados dentro da aplicação.

3.1.5 Entrada/Saída de Comunicações

Entrada de Comunicações

Para além da comunicação direta com o sistema é possível interagir com o sistema através de canais externos, por exemplo, email. Assim, torna-se possível a um utilizador abrir Pedidos de Serviço sem ter que entrar diretamente no sistema. Esta operação é sustentada pela versão base do sistema o que simplifica o processo, sendo apenas necessário enviar um email para o endereço email fornecido pela *Oracle*. No entanto, para o cliente é importante existirem mais canais de interação. Por exemplo, um email nacional e outro internacional. Para isso, é necessário criar essas contas de email, e adicionar-lhes uma regra de reencaminhamento dos emails recebido para o email fornecido da *Oracle*. Para além disso, é necessário criar os respetivos canais de entrada para os emails criados, como é possível verificar abaixo (Figura 12).

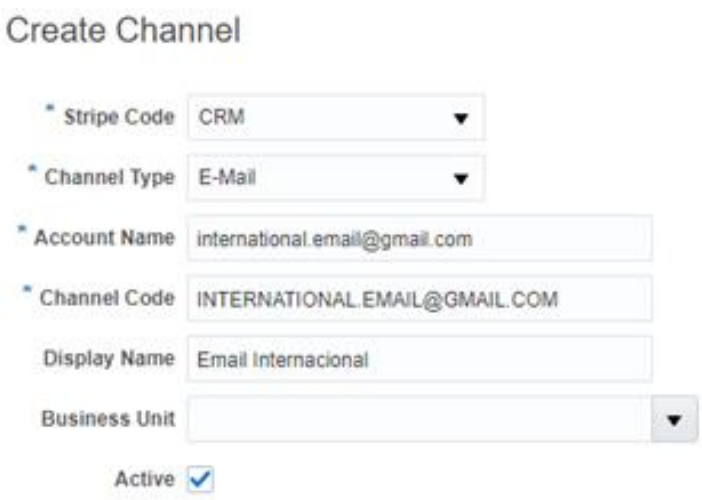

Figura 12. Menu de criação de um canal de comunicação

Assim, permitimos a identificação dos diversos canais de entrada na criação dos Pedidos de Serviço. Ou seja, quando um utilizador envia um email para o endereço "international.email@gmail.com" essa caixa de correio reencaminha a mensagem para o endereço dedicado do sistema e este faz a criação do Pedido de Serviço.

Saída de Comunicações

A saída de comunicações, por email ou SMS, é feita através de "*Workflows*". Um "*Workflow*" é semelhante a um *trigger*, mas permite algumas funções "*Standard*" especificas, como o envio de emails. Primeiro é necessário definir os "*Templates*" para os vários cenários em que um email será enviado, por exemplo e como podemos ver na imagem em baixo (Figura 13), um email para notificar o utilizador da criação do seu Pedido de Serviço.

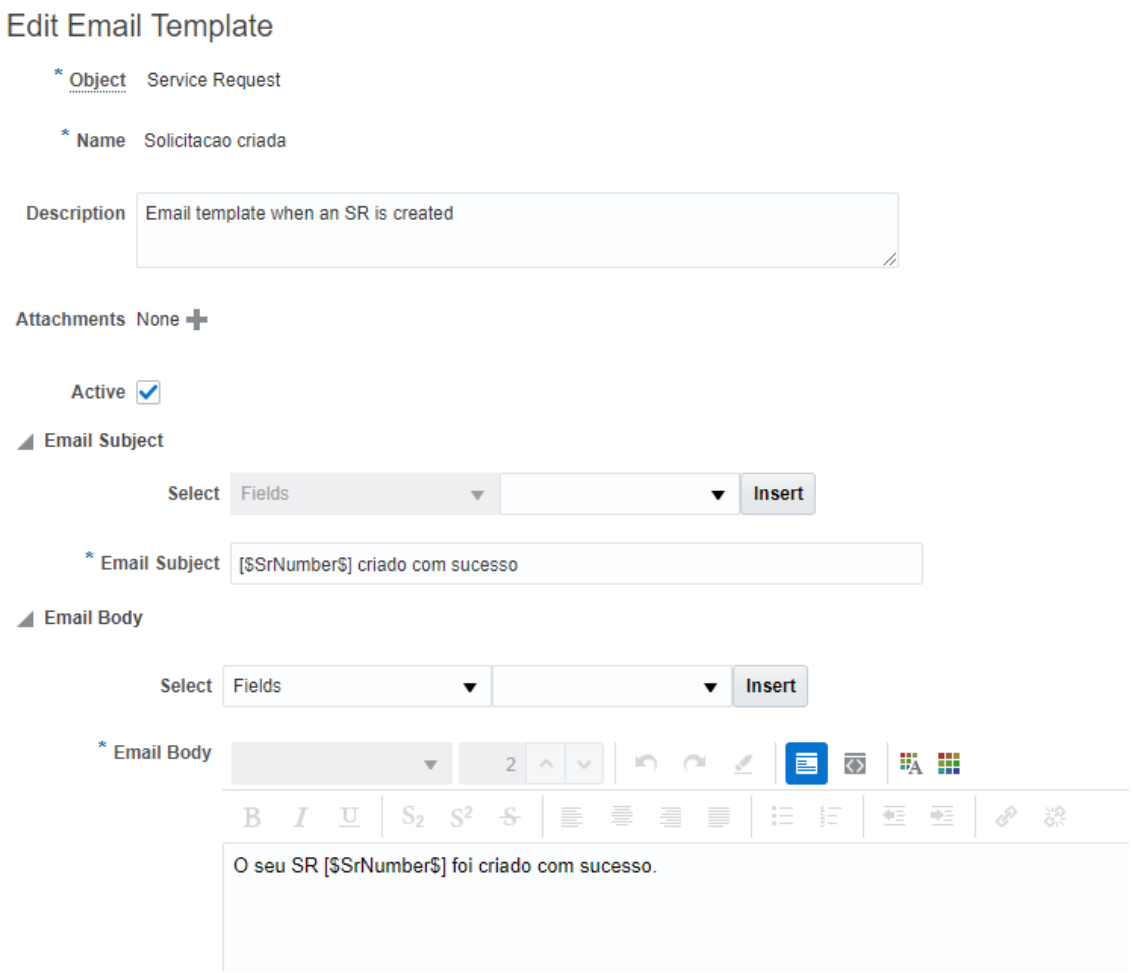

Figura 13. Ecrã de edição de um template de email

Após definido o modelo de email é criado o "*Workflow*" que o vai encaminhar Figura 14). Um "*Workflow*" é ativo apenas quando a sua condição é verdadeira, sendo que essa condição é avaliada quando um documento é criado e/ou atualizado. Se a condição se verificar, então as tarefas definidas no "*Workflow*" são efectuadas, neste caso o envio de um email, para o endenreço definido. Na imagem seguinte podemos ver uma condição exemplo no "*Workflow*", e em seguida a definição do email, onde se escolhe o "*Template*" de email a enviar, e todos os campos normais de um email, em particular o endereço de destino, que neste caso é importado diretamente de um campo no sistema.

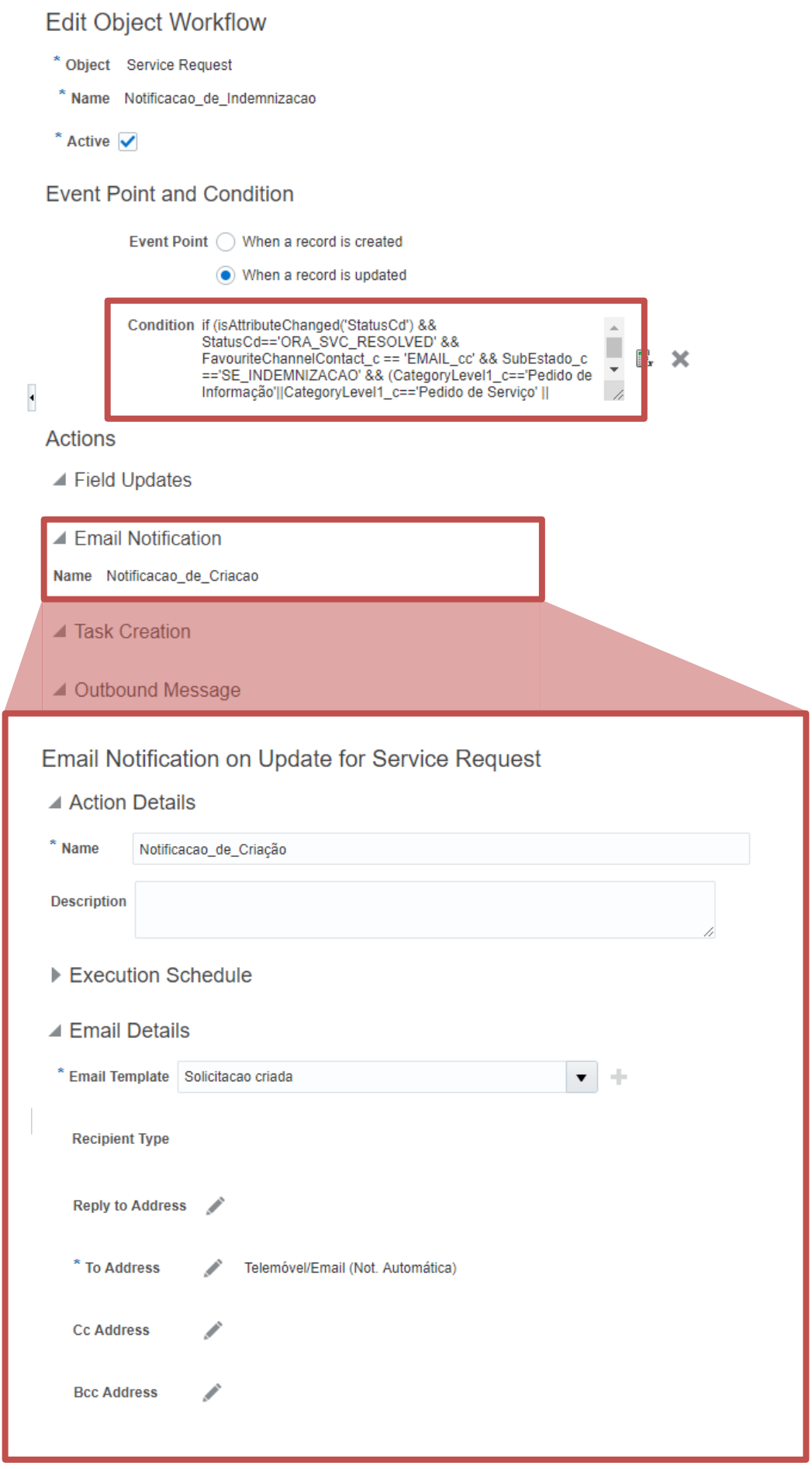

Figura 14. Processo de criação de um Workflow para o envio de uma notificação via email

3.1.6 Segurança e Perfis de Utilizadores

O acesso às áreas do *Engagement Cloud* é restringido e controlado por um sistema de políticas de segurança e permissões. Por exemplo, um utilizador pretende aceder aos Pedidos de Serviço ou ao separador Contas. Para isso, o utilizador irá necessitar de "*Roles*" específicos com permissões de acesso a esses objetos. Na imagem em baixo (Figura 15) está um exemplo de criação de um perfil com permissões para importação massiva de objetos.

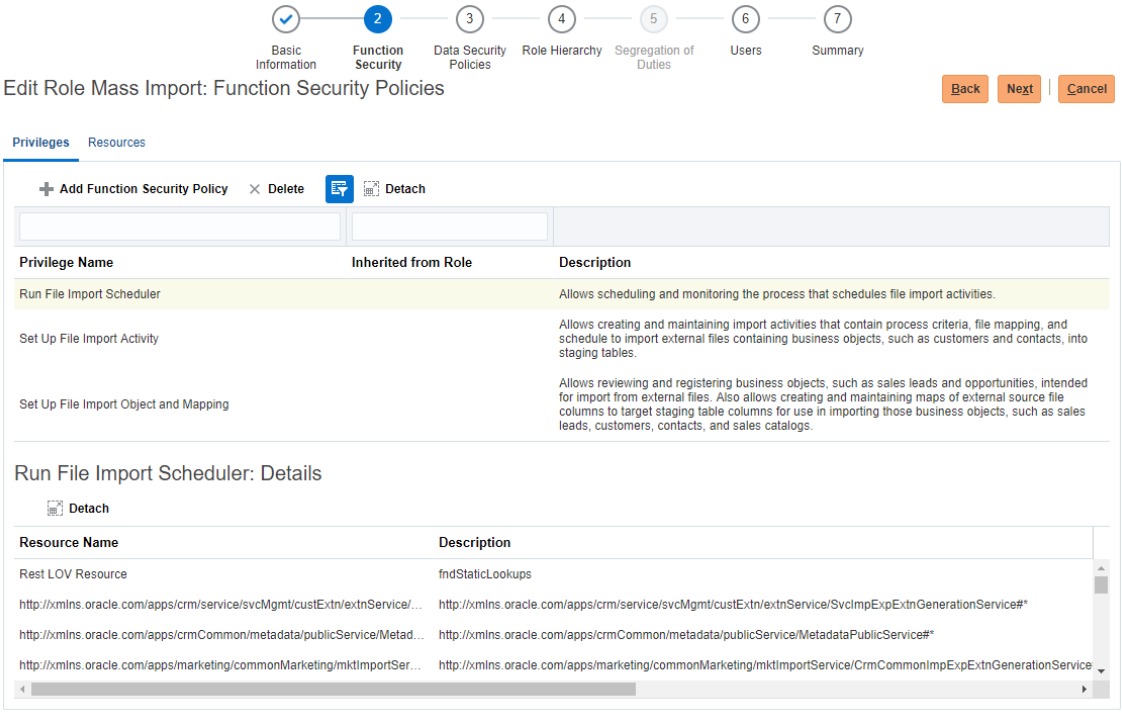

Figura 15. Menu de edição de um perfil de segurança, mais concretamente no separador das políticas de segurança de funções.

Primeiro definem-se as "*Function Security Policies*", ou seja, as políticas de segurança para determinadas funções ou operações na aplicação. No caso do exemplo, e caso que será abordado no capítulo imediatamente a seguir, para importações massivas de objetos, atribuíram-se as políticas de segurança que permitem o acesso às funções de Importação ("*Run File Import Scheduler*", "*Set Up File Import Activity*", "*Set Up File Import Object and Mapping*"). Estas três políticas incluídas no "*Role*" permitem que o utilizador possa agendar uma atividade de importação massiva, que possa efetivamente fazer uma importação massiva e que possa mapear os campos de um ficheiro *CSV* para os respetivos campos dentro do *OEC*.

Em seguida, na definição do perfil de importações, é necessário conceder-lhe políticas de segurança de acesso aos dados (Figura 16). Neste caso, atribuíram-se políticas de acesso aos objetos da componente "*Sales*" do *CRM* (Contas, Contactos, Oportunidades, etc.) e aos objetos da componente "*Service*" (Pedidos de Serviço, Filas, etc.).

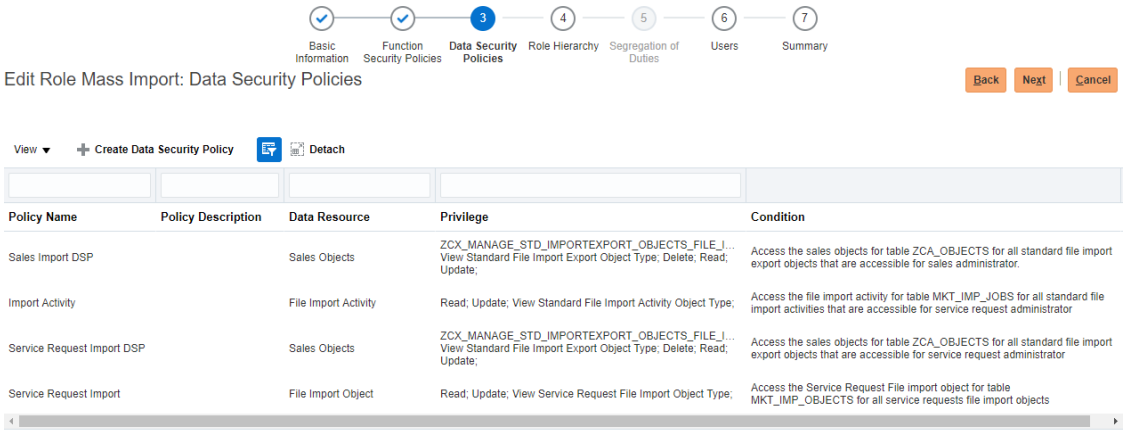

Figura 16. Menu de edição de um perfil de segurança, mais concretamente no separador das políticas de segurança de dados.

Nesta criação é possível definir as hierarquias de perfis, isto é, em que posição este perfil se vai situar, se será um perfil independente ou um perfil que se inclui noutro.

Por fim, podemos indicar para que utilizadores este "*Role*" estará ativo, por outras palavras, indicar quais os utilizadores com o perfil para importações em massa.

3.1.7 Unidades de Negócio

As unidades de negócio, ou *Business Units (BU)*, são segmentações do sistema em componentes de negócio, isto é, para cada setor de atividade do cliente, foram estruturadas diferentes unidades de negócio, dez, para ser mais preciso. Para cada uma delas, existem então configurações específicas, como por exemplo, diferentes categorias, diferentes canais de entrada, *templates* de email, *milestones*, etc. Sendo que todas as configurações mencionadas têm sempre em conta qual a unidade de negócio a que se destinam as configurações. No caso das importações, como as categorias ou os produtos, é necessário também indicar qual a *BU* à qual dada categoria ou dado produto ficaram afetos (Figura 18).

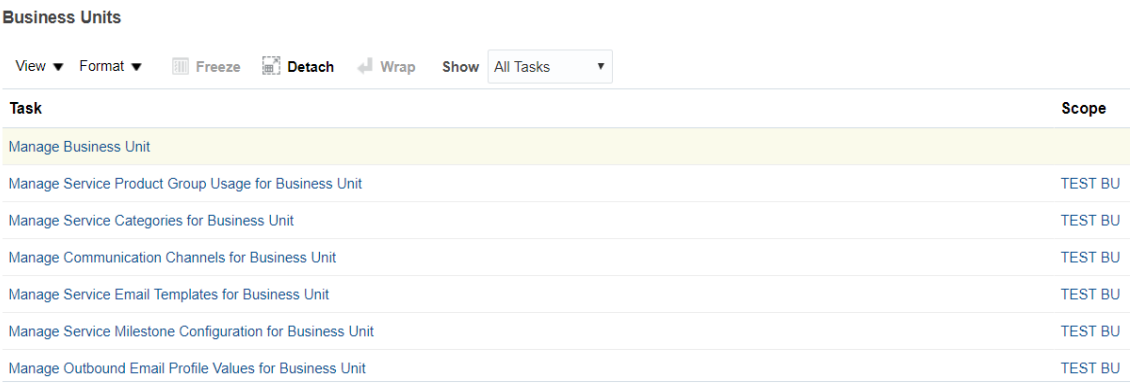

Figura 17. Visão do Setup & Maintenance sobre as Business Units

A criação de uma *Business Unit* tem apenas quatro parâmetros, sendo que apenas dois são de preenchimento obrigatório, o "Nome" e o "*Default Set*". O "*Default Set"* é a designação da segmentação feita no ambiente para esta unidade. O campo "*Manager*" serve para identificar qual o possível gestor da *BU* que pretendemos criar (Figura 17). Para poder ser elegível para o cargo de "*Manager*" de uma *BU*, o utilizador tem que possuir perfis de segurança com permissões e configurações de "*Manager*", como por exemplo os *roles* de "*Sales Manager*" ou "*Service Manager*". Estes *roles*, conferem a um utilizador controlo total sobre todos os objetos do espectro do "*Sales*" e do "*Service*" respetivamente. Por fim, o campo da localização refere-se à localização física real, definida em sistema para esta unidade de negócio.

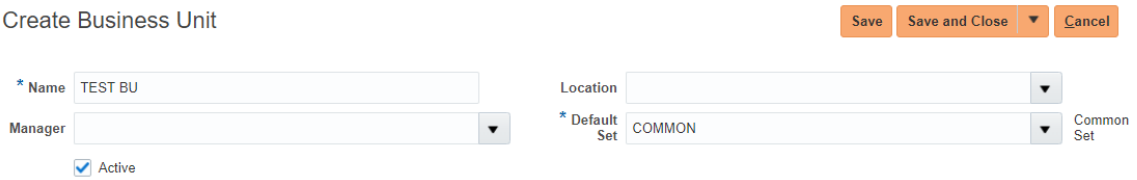

Figura 18. Encrã de criação de uma Business Unit

3.1.8 Importação Massiva de Objetos

A importação massiva de objetos, ou melhor, a importação massiva de dados de uma entidade, é algo essencial para o sistema, uma vez que são estes dados que vão popular os atributos e permitir as operações de negócio. O processo de importação massiva passa sempre por um processo semelhante entre as diversas entidades. Por este motivo, falemos primeiro do processo em si seguindo o exemplo da importação de categorias.

A lista de categorias requisitadas pelo cliente é extensa, o que torna a criação de categorias uma por uma, e manualmente, um trabalho moroso e exaustivo. Deste modo, a forma mais eficiente e rápida de o fazer é importando-as massivamente. A importação

massiva é possível para qualquer objeto dentro do *OEC*, por exemplo Pedidos de Serviço, Contas, Contactos, etc.

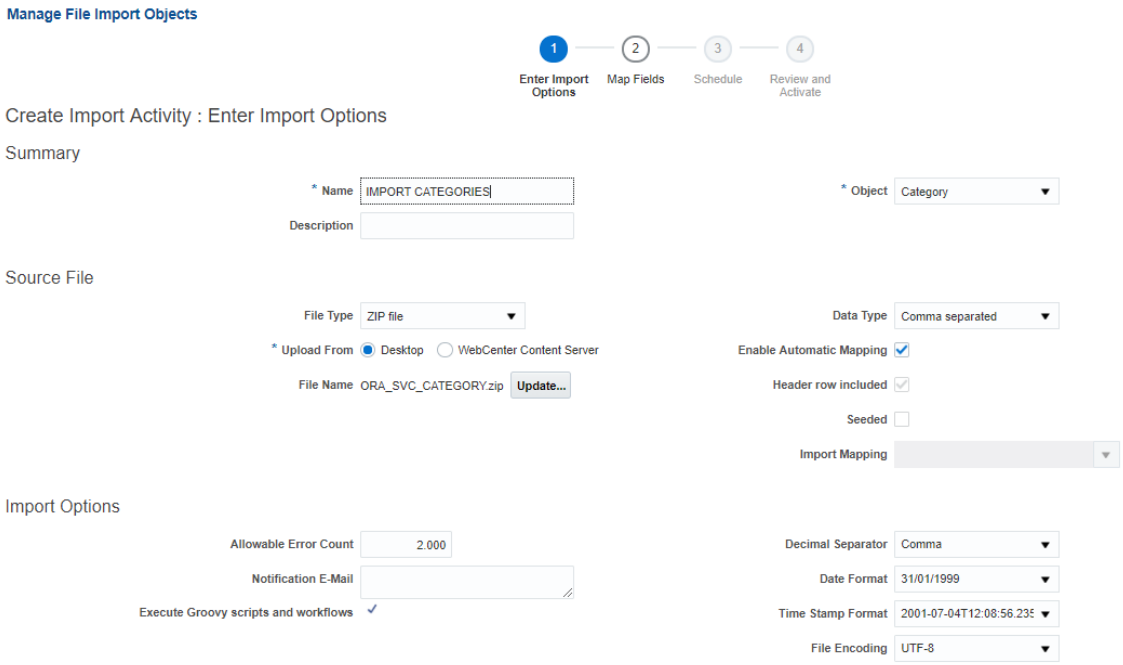

Figura 19. Ecrã de gestão de importações, mais concretamente o ecrã de opções

No "*Setup and Maintenance*", a área de trabalho onde se gerem e definem as propriedades globais do sistema, aceder à configuração "*Manage Import Activities*",

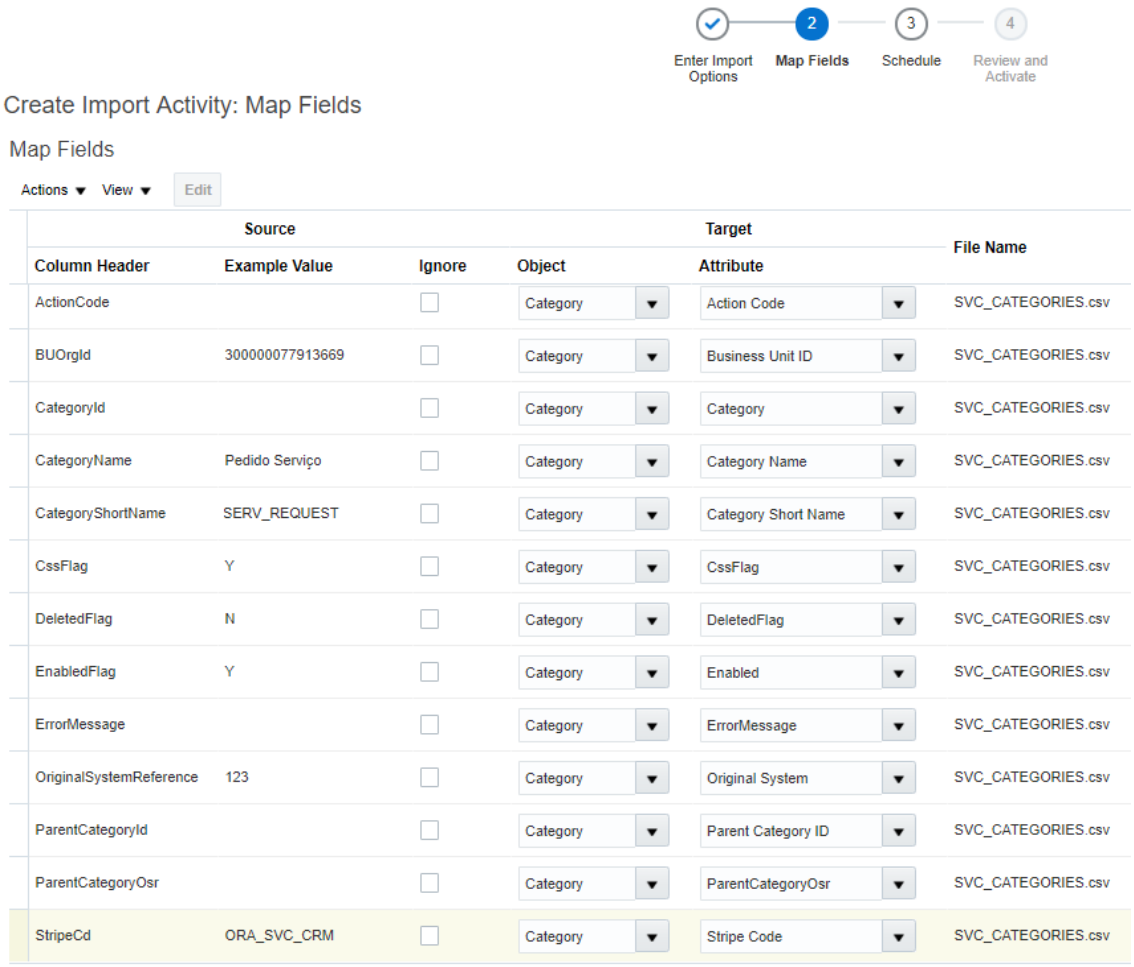

Figura 20. Mapeamento das colunas do ficheiro CSV com os campos existentes na base de dados do sistema

onde é possível agendar uma nova atividade de importação. Nesse menu, ao criar uma nova importação, podemos escolher qual o objeto, neste caso, o "*Category*" e de seguida escolher o ficheiro Zip com o nome específico "*ORA_SVC_CATEGORY.zip*", onde está o ficheiro *CSV*/Excel com os dados a importar (Figura 19). É ainda necessário definir o delimitador de campos, neste caso, a virgula. O sistema faz automaticamente o mapeamento dos campos no ficheiro *CSV* para os campos existentes do objeto escolhido (Figura 20). Por fim, a importação é agendada e confirmada, sendo efetuada na data e hora definida.

Quanto ao ficheiro *CSV*, para o caso das categorias em particular é definido com os campos necessários para definir e criar uma categoria. O nome da categoria e o código, e a referência interna no sistema, que é utilizada para definir qual a relação hierárquica entre as diversas categorias. Ou seja, qual o "pai" de uma determinada categoria, que é definido preenchendo a coluna "*ParentCategoryOsr*" com o valor de referência em sistema da categoria de nível acima (Figura 21).

A	B	C	D	E	F	G	H	I	J	K	L	M
ActionCode	BUOrgId	CategoryId	CategoryName	CategoryShortName	CssFlag	DeletedFlag	EnabledFlag	ErrorMessage	OriginalSystemReference	ParentCategoryId	ParentCategoryOsr	StripeCd
	300000077913669		Pedido Serviço	SERV_REQUEST	Y	N	Y		123			ORA_SVC_CRM
	300000077913669		Recolha	PICKUP	Y	N	Y		456		123	ORA_SVC_CRM
	300000077913669		Alteração recolha	CHANGE_PICKUP	Y	N	Y				456	ORA_SVC_CRM
	300000077913669		Anulação recolha	CANCEL_PICKUP	Y	N	Y				456	ORA_SVC_CRM
	300000078143244		Pedido Serviço	SERV_REQUEST	Y	N	Y		321			ORA_SVC_CRM
	300000078143244		Recolha	PICKUP	Y	N	Y		654		321	ORA_SVC_CRM
	300000078143244		Alteração recolha	CHANGE_PICKUP	Y	N	Y				654	ORA_SVC_CRM
	300000078143244		Anulação recolha	CANCEL_PICKUP	Y	N	Y				654	ORA_SVC_CRM

Figura 21. Ficheiro CSV exemplo para importação massiva de categorias

Este ficheiro é apenas um exemplo, na realidade, o ficheiro de importação de categorias foi dividido em dez ficheiros distintos, um por cada unidade de negócio, sendo que no total foram importadas cerca de mil e trezentas categorias distribuídas pelas diferentes dez *Business Units*. Estes ficheiros foram produzidos com base num ficheiro Excel fornecido pelo cliente, onde estavam todas as categorias distribuídas em cinco níveis e associadas às respetivas unidades de negócio. Dada a quantidade de categorias, e consequentemente, linhas de Excel a trabalhar, foi tomada a opção de gerar um código ("*CategoryShortName*") sequencial que ficou também igual ao código de referência do sistema ("*OriginalSystemReference*"), algo fácil de alcançar em Excel. O mais complexo de tudo isto é relacionar corretamente cada categoria, de nível superior a dois, com as suas respetivas categorias "Pai" ("*ParentCategoryOsr*"). Os códigos incluem as iniciais da *Business Unit*, concatenadas com um valor sequencial cujo primeiro algarismo indica o nível a que a categoria pertence (Figura 22).

A	B	C	D	E	F	G	H	I	J	K	L	M
ActionCode	BUOrgId	CategoryId	CategoryName	CategoryShortName	CssFlag	DeletedFlag	EnabledFlag	ErrorMessage	OriginalSystemReference	ParentCategoryId	ParentCategoryOsr	StripeCd
	300000000859008		Artigos Proibidos	CO_40001	Y	N	Y		CO_40001		CO_30042	ORA_SVC_CRM
	300000000859008		Documentos	CO_40002	Y	N	Y		CO_40002		CO_30042	ORA_SVC_CRM
	300000000859008		Encargos Aduaneiros	CO_40003	Y	N	Y		CO_40003		CO_30042	ORA_SVC_CRM
	300000000859008		Estado Processo	CO_40004	Y	N	Y		CO_40004		CO_30042	ORA_SVC_CRM
	300000000859008		Pedido 2ª Notificação	CO_40005	Y	N	Y		CO_40005		CO_30042	ORA_SVC_CRM
	300000000859008		Suspeita Contrafação	CO_40006	Y	N	Y		CO_40006		CO_30042	ORA_SVC_CRM
	300000000859008		Encargos Aduaneiros	CO_40007	Y	N	Y		CO_40007		CO_30043	ORA_SVC_CRM
	300000000859008		Estado Processo	CO_40008	Y	N	Y		CO_40008		CO_30043	ORA_SVC_CRM
	300000000859008		Encargos Aduaneiros	CO_40009	Y	N	Y		CO_40009		CO_30044	ORA_SVC_CRM
	300000000859008		Estado Processo	CO_40010	Y	N	Y		CO_40010		CO_30044	ORA_SVC_CRM
	300000000859008		Encargos Aduaneiros	CO_40011	Y	N	Y		CO_40011		CO_30045	ORA_SVC_CRM
	300000000859008		Dúvidas Utilização	CO_40012	Y	N	Y		CO_40012		CO_30046	ORA_SVC_CRM
	300000000859008		Report Erros	CO_40013	Y	N	Y		CO_40013		CO_30046	ORA_SVC_CRM
	300000000859008		Esclarecimentos	CO_40014	Y	N	Y		CO_40014		CO_30056	ORA_SVC_CRM
	300000000859008		Pedido de Apoio	CO_40015	Y	N	Y		CO_40015		CO_30056	ORA_SVC_CRM
	300000000859008		Esclarecimentos	CO_40016	Y	N	Y		CO_40016		CO_30061	ORA_SVC_CRM
	300000000859008		Pedido de Apoio	CO_40017	Y	N	Y		CO_40017		CO_30062	ORA_SVC_CRM
	300000000859008		Esclarecimentos	CO_40018	Y	N	Y		CO_40018		CO_30071	ORA_SVC_CRM
	300000000859008		Pedido de Apoio	CO_40019	Y	N	Y		CO_40019		CO_30072	ORA_SVC_CRM
	300000000859008		Esclarecimentos	CO_40020	Y	N	Y		CO_40020		CO_30076	ORA_SVC_CRM
	300000000859008		Pedido de Apoio	CO_40021	Y	N	Y		CO_40021		CO_30077	ORA_SVC_CRM

Figura 22. Excerto de um dos ficheiros CSV de importação

3.1.9 Utilização das Categorias

Na sequência da secção anterior, relativa à importação das categorias, é importante falar também de qual a sua utilização e impactos no *OEC*. As categorias são um dos pilares da componente *Service* na medida em que, todos os *Service Requests* criados necessitam de ser classificados ou categorizados. Além disto, as categorias são a base de todas as regras de negócio, e representam a maior fatia das regras e configurações necessárias. Em geral, todos os cenários, fluxos e customizações feitos começam por verificar qual a categoria escolhida, e em seguida aplicam as necessárias funções de

negócio. Por outras palavras, o que isto significa é que as categorias estão implicadas em grandes quantidades de código.

Posto isto, e sabendo que os nomes das categorias podem ser alterados, existe uma grande possibilidade de conflito que inutilize todas condições previamente definidas onde esses nomes estavam a ser utilizados. Assim, definiu-se uma função que obtém os códigos das categorias (que nunca são alterados pois são imutáveis) por nível. Isto é, se quisermos obter o código de uma categoria de terceiro nível, chamamos a função "*getCatLevelSC(3)"* que irá devolver o código da categoria desejada. Assim, é possível alterar o nome das categorias sem nunca interferir com condições ou validações. Esta função é também necessária uma vez que os códigos apenas são acessíveis por W*eb Service* e nunca de forma direta da aplicação.

```
if(size == 0 || CatID == null){
  return CatLvlsShortCode;
}

 def conn = adf.webServices.CategoryHierarchyWebservice
 def data = null
 try{
    data = conn.GET(CatID.toString())

    CatLvlsShortCode = data['CategoryShortName']+ "/" + CatLvlsShortCode;
    CatID = data['ParentCategoryId'].toString();

   }catch (e) {
     throw new oracle.jbo.ValidationException('Error in Category Hierarchy Short Code Generation  ' + e.toString())
   }

return getCatSC(size-1,CatLvlsShortCode,CatID);
```

Figura 23. Excerto do código Groovy para a obtenção dos códigos das categorias

No código acima, na figura 23, é utilizado um *Web Service* que acede ao objeto categoria com o *ID* da categoria selecionada, e percorre recursivamente toda a hierarquia acima dessa categoria, e para cada nível acedido concatena o valor do código para um campo *String*. Para efetivamente utilizar este código é necessário chamar a tal função "*getCatLevelSC(LEVEL)*" que dado o nível separa a *String* com os códigos concatenados e devolve-o. Deste modo, em todas as configurações e regras aplicadas, a identificação de uma categoria é feita pela chamada a essa função.

3.1.10 *Milestones, Entitlements* e *SLAs*

Estes conceitos são centrais no contexto da "vida" de um Pedido de Serviço. As *Milestones* referem-se aos principais marcos no processamento de um Pedido, ou seja, os acontecimentos que têm maior impacto no seu ciclo de tratamento. Por exemplo, o marco entre o registo de um Pedido e a sua entrada em resolução, ou, o marco entre a entrada em resolução e a sua conclusão. Estes marcos surgem geralmente associados aos chamados *SLAs*, ou "*Service Level Agreements*". Estes *SLAs* são métricas temporais que limitam o alcance de uma *Milestone*. Por exemplo, uma métrica de 2 dias úteis para

que um Pedido de Serviço seja resolvido. É então a este nível que surge a relação entre *Milestones* e *SLAs*, onde cada métrica temporal definida por um *SLA* fica associada a uma regra verificada por uma *Milestone*. Na generalidade dos casos, as *Milestones* e os *SLAs* são associados através das chamadas *Entitlement Rules*, que no fundo verificam se as condições das *Milestones* foram atingidas e gere a contagem do tempo dos *SLAs*.

Isto é relevante para o utilizador, mais propriamente, um operador ou agente responsável por resolver os Pedidos de Serviço, pois define quanto tempo tem para atingir as diversas etapas do processo. E é também relevante para quem gere os responsáveis uma vez que se torna possível verificar quais os tempos de resolução e se estes superam os limites estabelecidos pelos *SLAs*.

Milestone Configuration

Ao nível do *Oracle Engagement Cloud* estas regras começam por ser definidas da mesma forma. Primeiro, definem-se as *Milestones* no *Setup and Maintenance* atribuindo os respetivos marcos de começo (1), pausa (2) e encerramento (3) (números relativos à Figura 24).

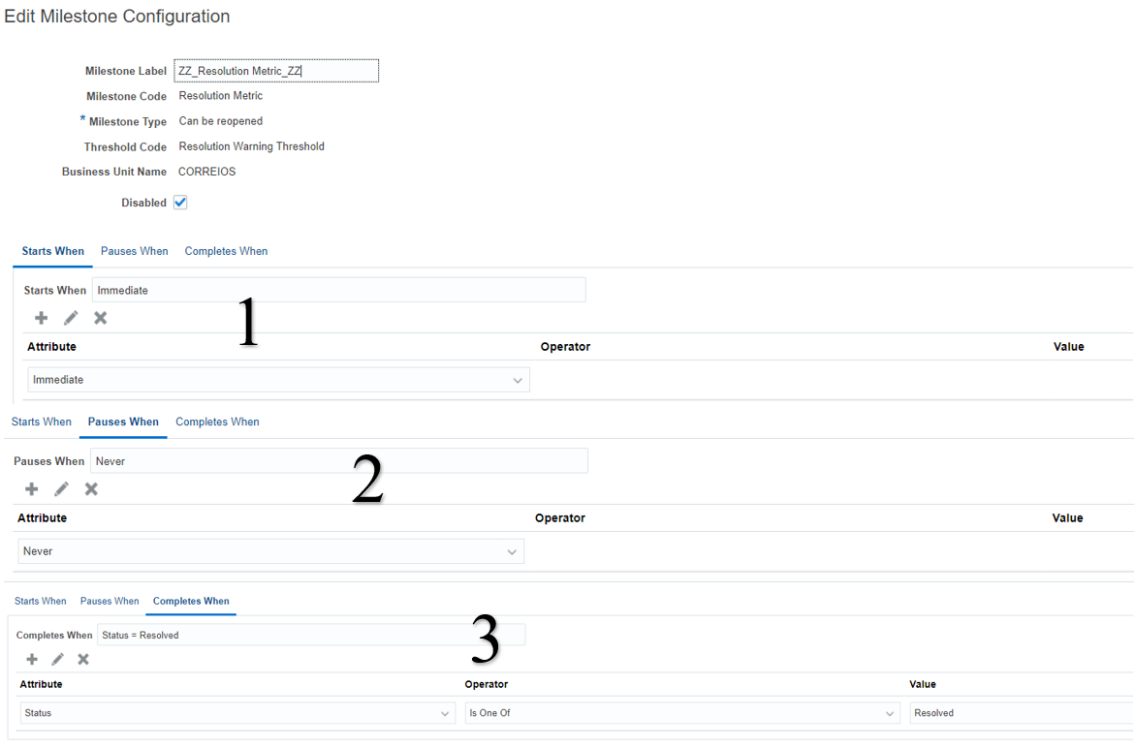

Figura 24. Processo de Criação/Edição de uma Milestone

Manage Service Mappings

De seguida é necessário seguir uma série de passos para preparar uma *Milestone* e um *Entitlement*. O primeiro destes passos passa por configurar os mapeamentos dos campos das entidades pretendidas, por exemplo, do *Service Request*. O mapeamento começa com a criação de atributos do lado dos *Entitlements* (Figura 25), e em seguida, criar as associações entre os atributos criados e os existentes na entidade (Figura 26). Por exemplo, criar o atributo "*Cat_1_c_Custom*" para relacionar com o campo "*CategoryLevel1_c*" da entidade *Service Request*.

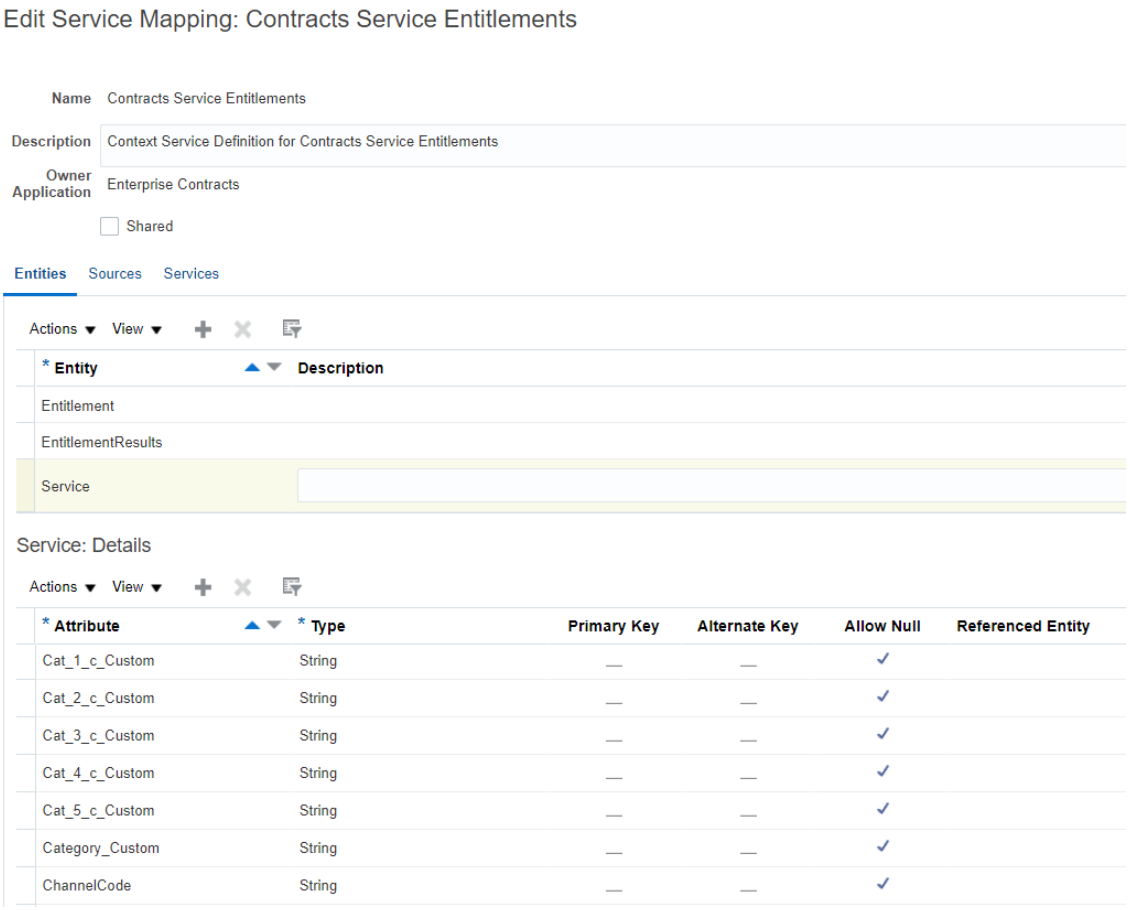

Figura 25. Ecrã de edição e criação dos campos para os Entitlements

Com isto, obtemos uma configuração dos campos que desejamos ter disponíveis para utilizar no passo seguinte, a configuração da Matriz de Condições.

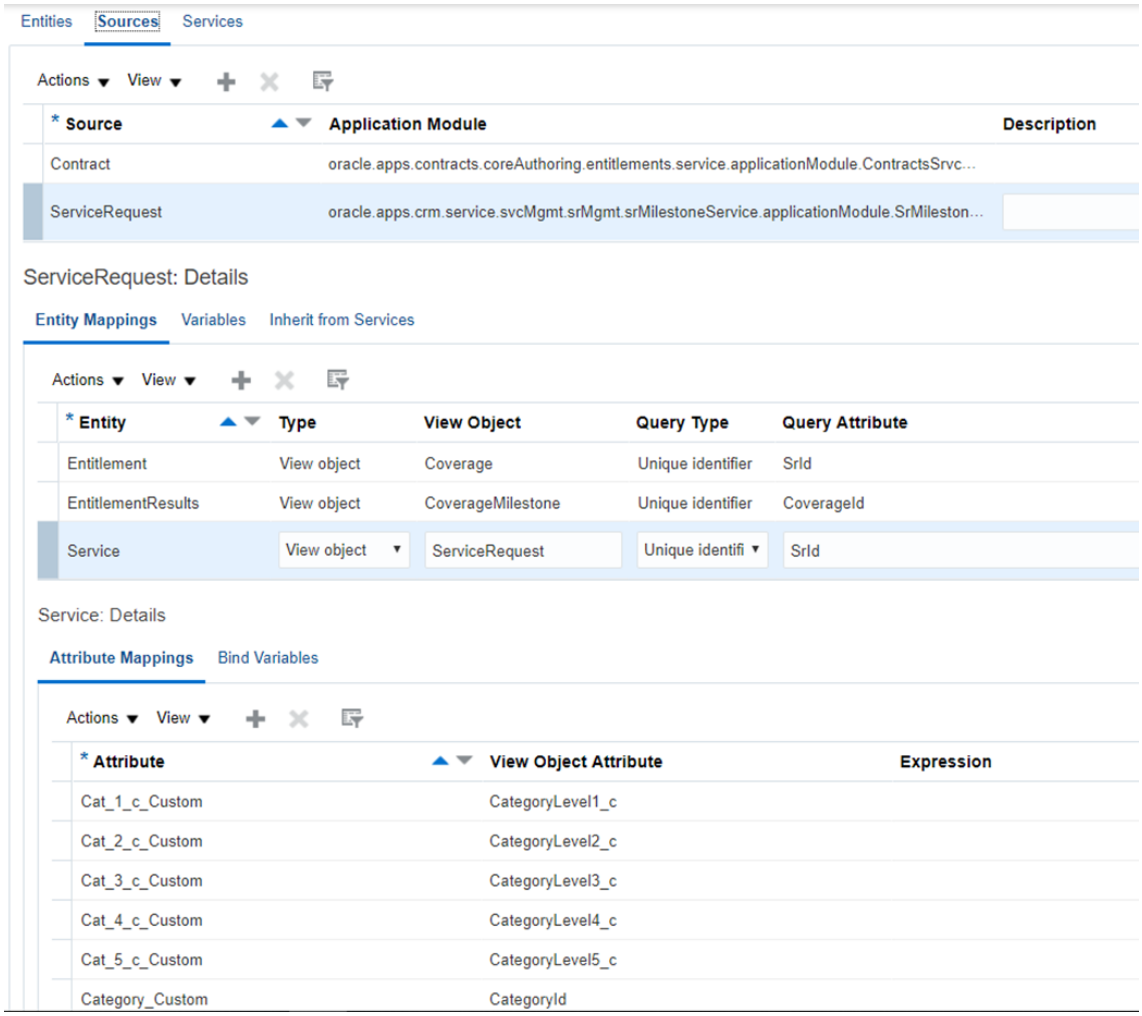

Figura 26. Ecrã de mapeamento dos campos do Entitlement criados com os atrinutos da Entidade Service Request

Manage Matrix Classes (Matriz de Condições)

Após definidos os atributos que pretendemos utilizar para os *Entitlements*, temos que definir quais as condições e resultados que se apliquem. Para preparar as condições utilizamos os atributos que definimos anteriormente no mapeamento e sobre estes aplicamos uma validação. Por exemplo, comparar se o atributo "*Cat_1_c_Custom*" é igual a um determinado valor de texto (Figura 27). Estas condições serão utilizadas mais à frente na definição final das regras.

Em seguida, ainda no mesmo contexto, definem-se também os resultados possíveis. Os resultados são na verdade as *Milestones* definidas logo no primeiro passo.

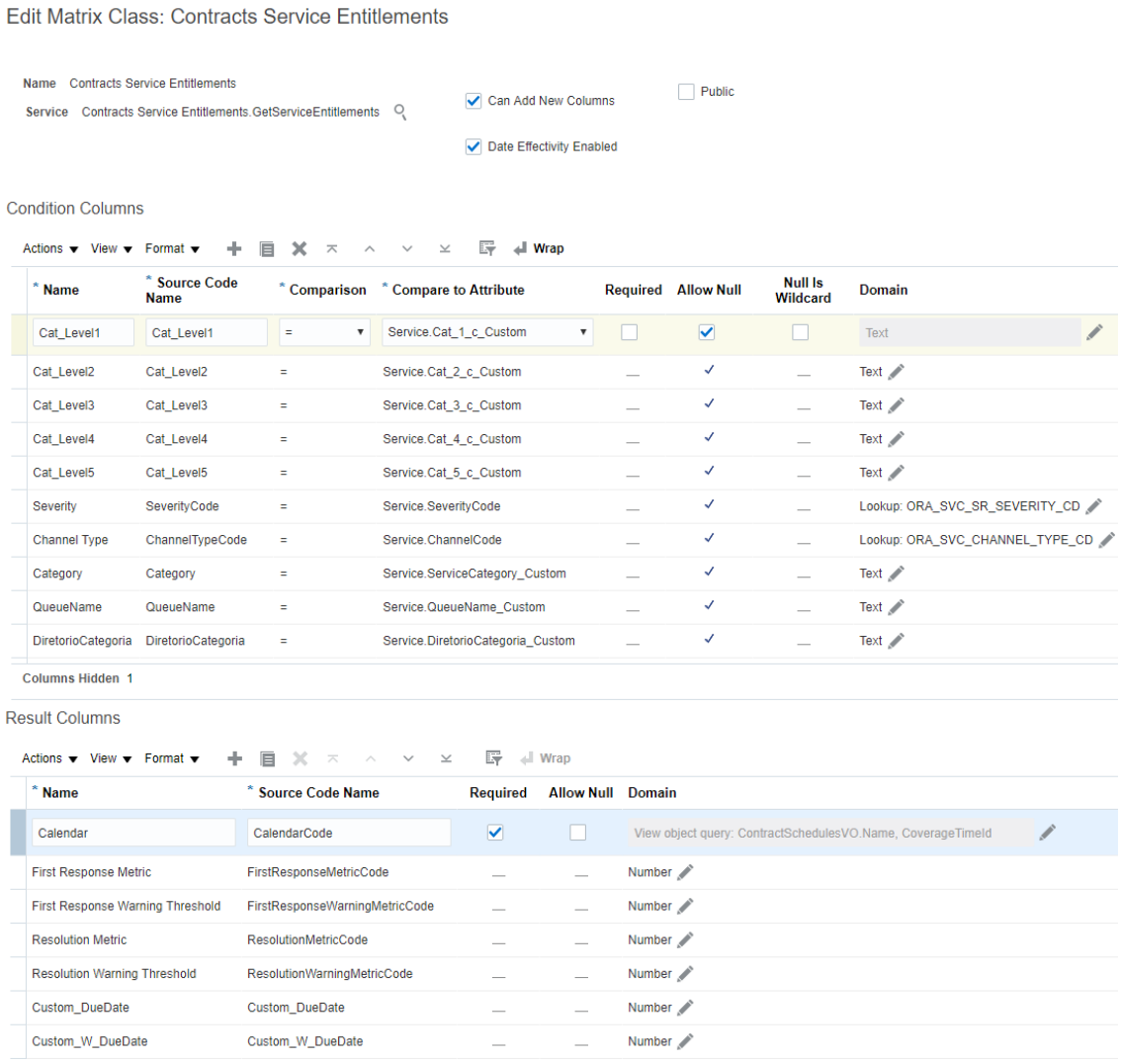

Figura 27. Ecrã de edição da Matriz de Condições

Deste modo, nesta etapa definimos quais as *Milestones* passíveis de serem utilizadas ao nível das regras dos *Entitlements* (*Entitlement Rules*).

Manage Availability

Em seguida, é necessário definir quais os horários de trabalho da pessoa responsável pelo Pedido de Serviço. Isto é essencial uma vez que determina quais os períodos de tempo em que o *SLA* irá correr. Por exemplo, o horário laboral de um dado operador é de oito horas, tendo início às nove horas da manhã e término às dezoito (Figura 28). Neste contexto, o *SLA* apenas poderá estar ativo durante os cinco dias úteis da semana e durante as oito horas de trabalho do agente, tendo ainda em consideração o intervalo para almoço. Assim, faz sentido que se definam todas estas condições de

atividade de um *SLA* para que o tempo estipulado para a conclusão de uma *Milestone* não seja contabilizado erradamente.

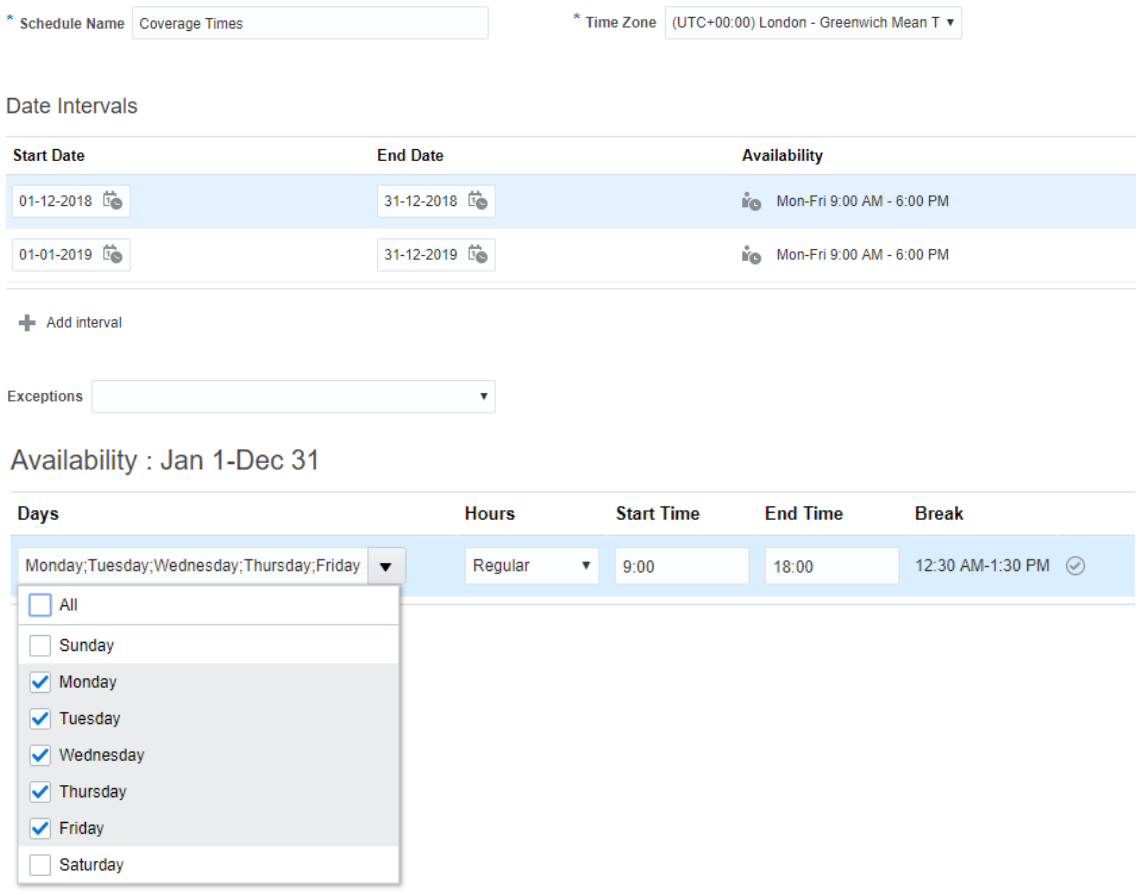

Figura 28. Ecrã de disponibilidade do operador, horários laborais

Neste momento temos já definidas as *Milestones*, as condições possíveis para despoletar as regras, os resultados, que são no fundo as próprias *Milestones* e os períodos de disponibilidade para a contagem dos *SLAs*. De seguida, temos então que definir as regras onde iremos aplicar as condições, os resultados, o valor para o *SLA* e os calendários de disponibilidade.

Standard Coverage

É neste passo que são definidas as *Entitlement Rules*. Ao adicionar uma nova regra selecionamos logo qual dos resultados pretendemos, resultado esse que diz respeito a uma *Milestone*, tudo definido nos passos um e três.

Estas regras têm três áreas principais, que têm relação direta com as configurações preparadas nas etapas anteriores. A primeira área contém as condições, onde utilizamos as condições que foram definidas no passo três, que por sua vez utilizam os atributos definidos no passo dois. Por exemplo, "*Cat_Level1 (=)*" igual a "Pedido Geral" valida se o primeiro nível da categoria escolhido é igual ao valor de texto "Pedido Geral".

Caso estas condições sejam validadas então a área seguinte pode prosseguir. Em seguida, a área referente aos resultados, definidos também na etapa três, e aos horários de aplicação das métricas de resolução (*SLAs*). O calendário refere-se ao horário definido no passo quatro e vai restringir a contagem temporal da métrica de resolução. Neste caso, definiu-se uma métrica de dez horas, ou seja, seiscentos minutos, o que significa que, o utilizador responsável terá dez horas, desde a abertura do pedido para o resolver, tendo em consideração o seu horário de trabalho. Por fim, a terceira área, que define o período de tempo no qual esta regra se irá aplicar.

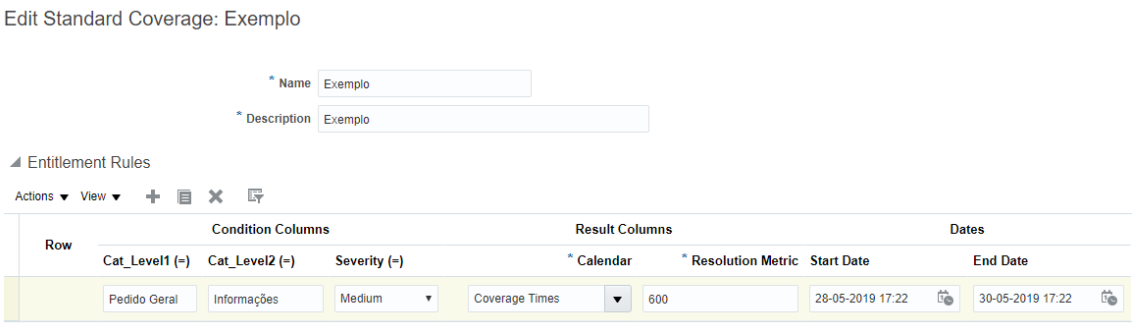

Figura 29. Ecrã de criação/edição das Entitlement Rules

Para o exemplo apresentado na figura 29, a leitura da regra criada é a seguinte:

Para um determinado Pedido de Serviço (*Service Request*), com a categoria "Pedido Geral - Informações" e com uma Severidade (*Severity*) de nível médio, deve ser aplicada uma métrica de resolução de dez horas, tendo em conta o calendário laboral dos operadores. Esta métrica é aplicada logo que o Pedido de Serviço é criado e apenas termina assim que o Pedido for resolvido, tal como definido na *Milestone*.

Default Coverage

Por fim, é necessário ativar a regra criada, para isso basta apenas definir quais as regras que queremos que se apliquem e para quem, se para toda a gente (Global) ou apenas para algumas contas. Neste caso vamos aplicar a nossa regra "Exemplo" para todos os utilizadores (Figura 30).

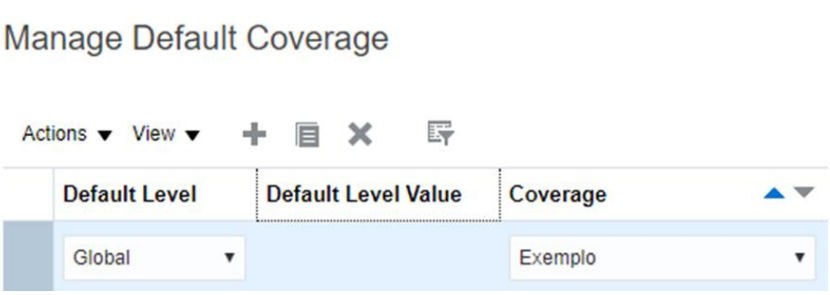

Figura 30. Menu de definição do alcance das regras (Entitlement)

3.1.11 *Queues* e *Service Requests Assigment Rules*

Quando um pedido de serviço é criado (*Service Request*) ele é automaticamente adjudicado a uma fila de trabalho. Uma fila de trabalho, ou "*Queue*", é essencialmente uma pilha de tarefas, onde os diversos pedidos de serviço são colocados para posteriormente serem associados a um operador/utilizador encarregado por o resolver.

A "*Queue*" é um objeto, e como tal é necessário criar instâncias desse objeto para associar os recursos (humanos) encarregados por essas filas de trabalho, e para definir as regras que determinam que pedidos de serviço pertencem a que filas (Figura 31).

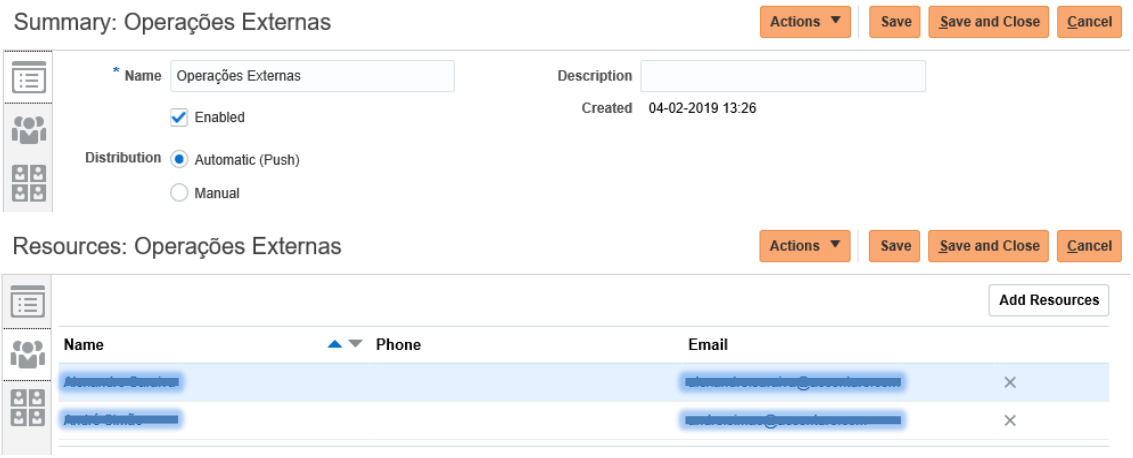

Figura 31. Ecrã de Criação/Edição de uma Queue

A criação de uma fila de trabalho ("*Queue*") passa por simplesmente pela atribuição de um nome e a seleção da opção "automática" para o caso de querermos associar os pedidos automaticamente. Após criada a fila de trabalho são adicionados os recursos responsáveis por tratar os pedidos de serviço que sejam adicionados a esta pilha de trabalho.

Para definir que um pedido de serviço pertence a uma determinada fila de trabalho são criadas regras e condições de associação/atribuição. Essas regras podem ser, por exemplo, "Se a categoria for igual a Recolha" (como se pode ver em baixo). Por outras palavras, sempre que for criado um "*Service Request*" com a Categoria igual a "Recolha" a fila de trabalho para onde será encaminhado esse pedido será "Operações Externas" (Figura 32).

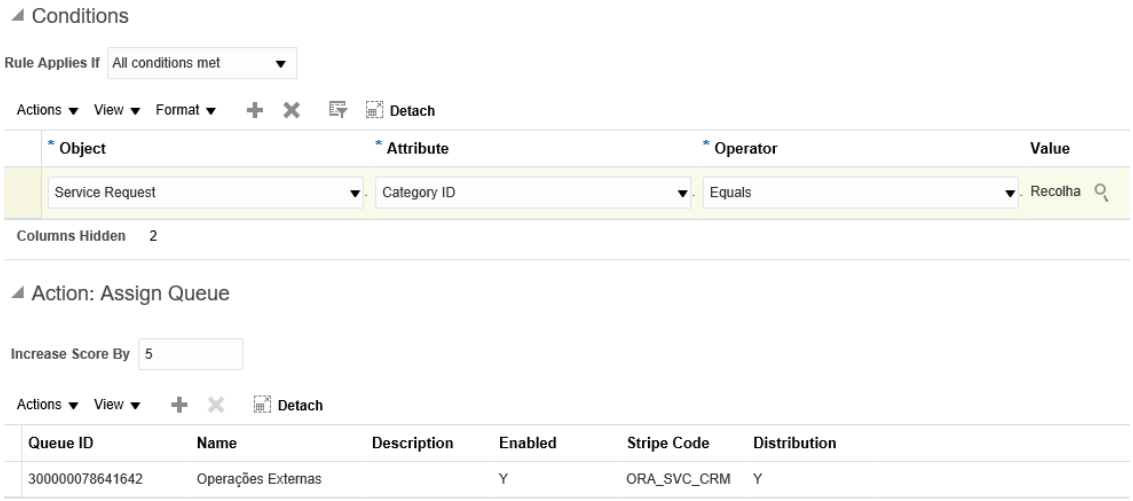

Figura 32. Ecrã de criação/edição de uma Regra de Encaminhamento

O *Oracle Engagement Cloud* é uma aplicação bastante completa para o propósito dos sistemas CRM, no entanto não cobre a totalidade das especificações. Por vezes os requisitos do cliente exigem uma customização da aplicação tal, que não é viável ou possível fazê-lo diretamente no *Engagement Cloud*. Deste modo, surge a necessidade de criar aplicações externas, e é com isto que passamos para o capítulo seguinte, onde vamos abordar o desenvolvimento de aplicações externas, complementares ao *OEC*, para cumprir alguns dos requisitos do cliente.

3.2 Aplicações Externas Complementares ao *OEC*

Como já referido, o *Engagement Cloud* não consegue ser solução para todas as situações e cenários requisitados pelo cliente. Como tal, é necessário colmatar essas falhas com funcionalidades e aplicações externas. Por este motivo, eu propus o desenvolvimento de uma aplicação *web* que, invocando uma *API* externa, fornecida pelo cliente, conseguisse povoar automaticamente os diversos campos de uma morada. De uma forma simples, o propósito desta aplicação era introduzir a funcionalidade de sugestão e *autocomplete* de moradas.

3.2.1 Interface

A interface da aplicação era um formulário com diversos campos necessários para a identificação de uma morada (Figura 33).

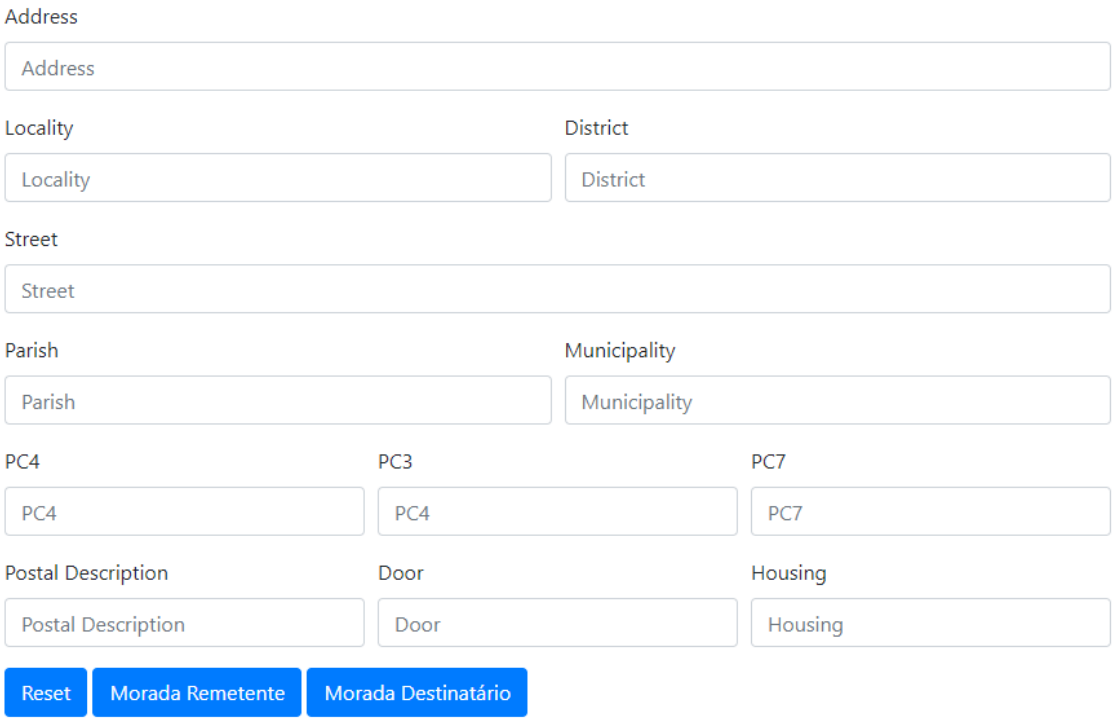

Figura 33. Interface da Aplicação Web Externa para recolha e "Autocomplete" de Moradas

O objetivo deste formulário é permitir ao utilizador aceder a uma funcionalidade de "*Autocomplete*" de moradas, algo que o *OEC*, à data da realização do presente relatório, não permite fazer nativamente. O formulário permite fazer uma pesquisa e filtragem de moradas, e seu posterior envio para o *OEC*, quer como "Morada Remetente", quer como "Morada Destinatário". Ou seja, o mesmo formulário pode servir para popular quaisquer campos referentes a moradas dentro do *OEC*, bastando para isso que se adicione um novo botão em que seja alterado o *JSON* onde se mapeiam os campos no destino, dentro o *Engagement Cloud*.

Este formulário foi feito em *HTML* com recurso à *framework* de *front-end Bootstrap*. Dada esta opção de desenvolvimento foi possível preparar este formulário em bastante pouco tempo.

3.2.2 Ligação à *API*

A componente fulcral desta aplicação é o "*Autocomplete*" das moradas, sendo esse o motivo pelo qual a aplicação foi desenvolvida. Para alcançar essa funcionalidade o cliente forneceu uma *API* que permite obter os dados de uma morada. A chamada a essa *API* é feita por *Javascript* através de um pedido *ajax*, do tipo *GET*, com os devidos

headers de autenticação, e como resposta é enviado um objeto *JSON*. Após obter esse *JSON* com a morada, é feito o mapeamento dos valores recebidos para os campos correspondentes no plugin da página, isto é, o script que recebe os dados do formulário dentro do *HTML*. Esse script permite recolher os dados dos campos do formulário, enviá-los para a função que liga à *API* e fazer um novo pedido *GET*. Ou seja, cada vez que um campo é alterado na página o script recolhe todos os valores preenchidos, passa-os para a função de chamada à *API*, que por sua vez efetua o pedido *GET*. Desse pedido é retornado um *JSON*, que é separado em campos conhecidos. Esses campos são então devolvidos à página e apresentados na lista de sugestões de seleção, como se pode observar na figura 38, apresentada mais à frente.

3.2.3 Reenviar os Dados para o *OEC*

Após contactar a *API*, obter os dados para sugestão e ter os valores apresentados no formulário precisamos de enviar os valores de volta para o *Engagement Cloud*. Este processo é conseguido através de uma nova chamada *REST*, desta vez, um pedido *PATCH*, que é acionado por um clique no botão de envio. Ora, tendo o *URL* base do *OEC* até á secção dos *Service Requests*, basta-nos saber qual o número específico do *Service Request* que pretendemos popular com os campos da morada. Esse número é passado no *URL* da chamada à aplicação. Deste modo, sabendo qual a instância do *Service Request* que queremos atualizar, podemos enviar os valores da morada num objeto *JSON*, com os atributos desse objeto a corresponderem aos vários campos da morada presentes no *OEC*. Ou seja, o objeto *Service Request*, no *OEC*, deve estar preparado com campos para receber cada um dos diferentes valores da morada (Rua, Código Postal, Distrito, Porta, etc.). Por fim, o pedido *Ajax* do tipo *PATCH* (assíncrono, de forma a correr em simultâneo com a página sem interferir com a sua apresentação) envia o objeto *JSON*, previamente produzido com os nomes dos campos no *OEC*, e atualiza-os.

3.2.4 Invocar a Aplicação dentro do *OEC*

Uma vez preparada a aplicação externa é necessária efetuar a ligação com o *Engagement Cloud*. Isto pode ser conseguido de duas formas, um link para a aplicação que abre um novo separador no browser ou, incorporando a aplicação dentro do *OEC*. No projeto foi implementada a segunda opção através de um "*Mashup*". Para criar um "*Mashup*" no *Engagement Cloud* é preciso registar a aplicação externa que pretendemos incorporar, e só depois chamar o "*Mashup*" na página pretendida. Na figura seguinte podemos ver o processo de registo de uma aplicação externa para um "*Mashup*", onde essencialmente é efetuado o registo do *URL* onde a aplicação se encontra alojada (Figura 34).

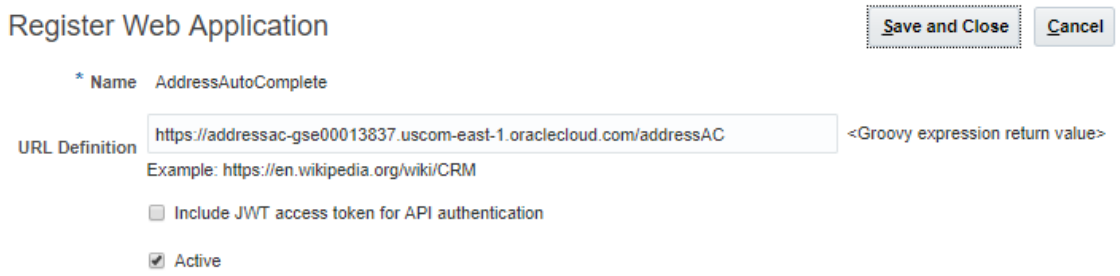

Figura 34. Registo de uma aplicação Externa no OEC

Depois de registada a aplicação num "*Mashup*" podemos chamá-lo para a página, neste caso, a página dos *Service Requests* onde temos os campos das moradas expostos. No contexto do *OEC*, um "*Mashup*" é um componente semelhante a uma *iFrame* para o *HTML*, por outras palavras, é uma moldura dentro da página recetora preparada para embeber e apresentar uma página externa. Na imagem seguinte (Figura 35) podemos ver como é inserido o "*Mashup*" na página dos *Service Requests*.

Primeiro escolhe-se a aplicação previamente registada com um *URL* base. Por fim, e como necessitamos de passar alguns parâmetros para a aplicação externa, adicionamos os campos necessários ao *URL*. Essa adição é feita por *script Groovy*, como é visível na figura em baixo, onde o campo "*SrNumber*" (número do *Service Request*) e o "*jwtToken*" são passados para o *URL*. O "*SrNumber*" é necessário para que a aplicação externa saiba qual a instância do *Service Request* que deve ser atualizada pelo *PATCH*. O "*jwtToken*" é uma medida de segurança comum para autenticação e para transferências de informação (Figura 35).

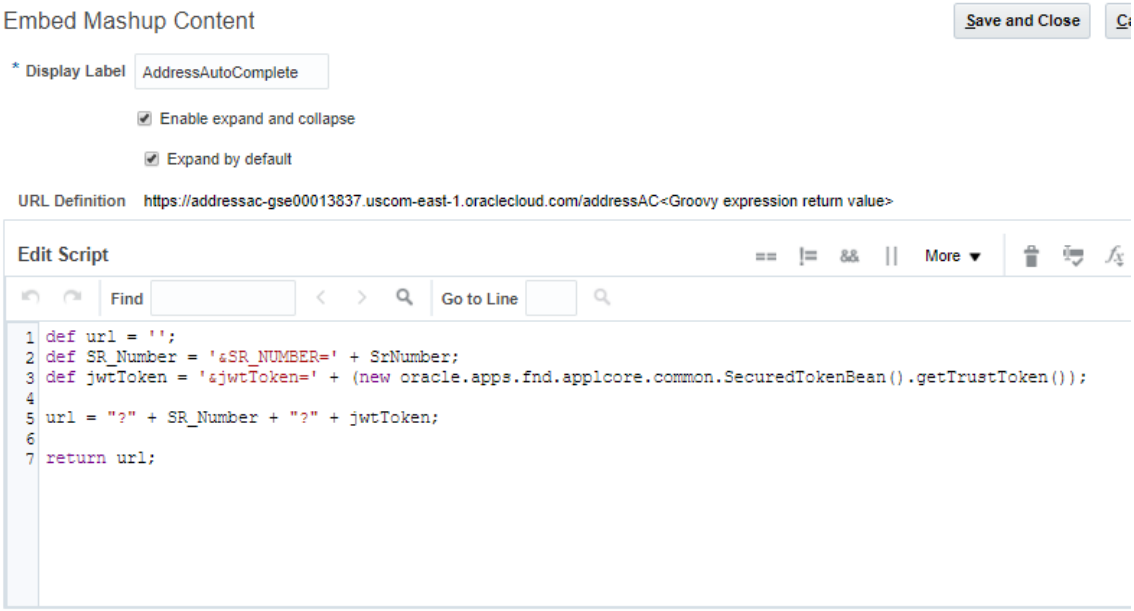

Figura 35. Invocação do Mashup para a página do Service Request

Por extenso, "*JSON Web Token*", é uma forma de transmitir informação de forma segura, entre duas aplicações, utilizando *JSON* encriptado dentro do *URL*. Neste caso, usamos os *JSON Web Tokens* para autorizar o regresso ao *OEC*, uma vez que uma das secções do "*jwt*" é gerada com base nas credenciais de acesso. No contexto da aplicação externa, o "*jwt*" é utilizado no cabeçalho "*Authorization*" do pedido *Ajax* com o esquema "*Bearer*" (*Authorization*: *Bearer* <*token*>). Assim, torna-se possível fazer pedidos entre aplicações diferentes e com domínios diferentes [3].

3.2.5 Evolução para "Dados Complementares"

No decorrer dos trabalhos do primeiro semestre foi desenvolvida uma aplicação *Web* para o preenchimento automático de moradas, com o nome "*Address Auto Complete*". Por uma questão de usabilidade e experiência de utilização, essa aplicação "evoluiu" para uma página *web* muito mais complexa, que passou a incluir a grande maioria dos campos extra do Pedido de Serviço. Esta alteração foi feita para evitar que o utilizador tivesse que alternar entre as diversas abas do Pedido de Serviço para poder gerar e/ou editar um registo, deste modo, toda a informação geral do pedido encontra-se num local único.

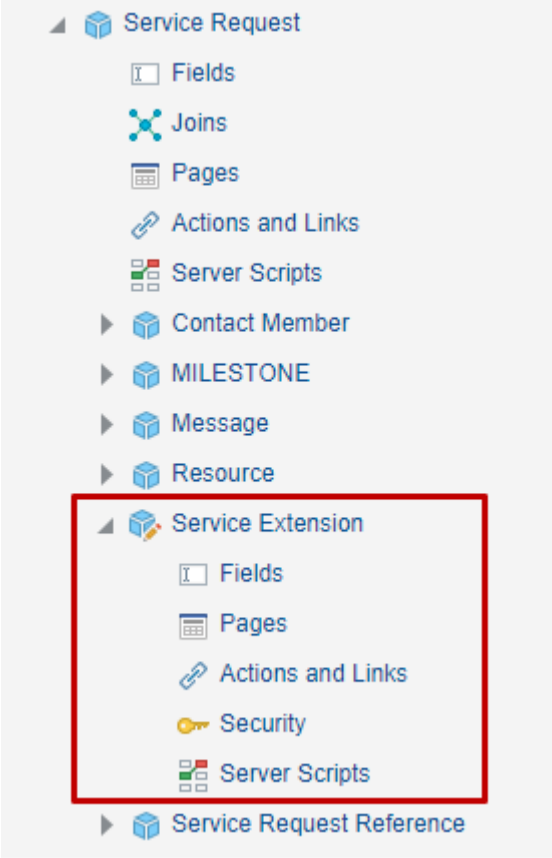

Figura 36. Menu do Application Composer na secção dos objetos, com destaque sobre a entidade Service Extension

A página tem duas funções principais, a de formulário de submissão de dados e a de visualização dos dados armazenados no *OEC* para aquele pedido. É também responsável por comunicar e integrar com *Web Services* do cliente. Para isto foi necessário criar uma extensão para o objeto "*Service Request*", ou seja, um novo objeto filho do objeto "*Service Request*" (*Child Object*), com o nome "*Service Extension*" (Figura 36). Esse novo objeto inclui todos os campos extra do pedido de serviço e serve, de certo modo, para armazenar os valores dos campos apresentados na página.

Para desempenhar as duas funções principais a aplicação usa as *APIs REST* do *Oracle Engagement Cloud*. Primeiro um *GET* para obter os valores guardados nos campos do "*Service Extension*" e popular o formulário com esses valores. Em segundo, um *PATCH* que é invocado quando o utilizador submete os dados preenchidos no formulário. A comunicação da página para o *OEC* é feita através do envio de objetos *JSON* com os valores dos campos, sendo que este *JSON* é gerado por *Javascript* na própria página, através da recolha de todos os valores preenchidos no formulário. Para o preenchimento dos campos, o processo é o inverso, uma vez que o método *GET* retorna também um objeto *JSON* com os campos e valores registados no *Service Extension*. Um dado importante para esta página é a garantia de existência prévia da instância (única) do *Service Extension* para um dado *Service Request*, pois só assim o método *PATCH* funcionará. Para isto, foi criada uma função, que é disparada no "*Trigger After Create*" do "*Service Request*", ou seja, sempre que um pedido de serviço for criado, será também criada a sua extensão única (Figura 37).

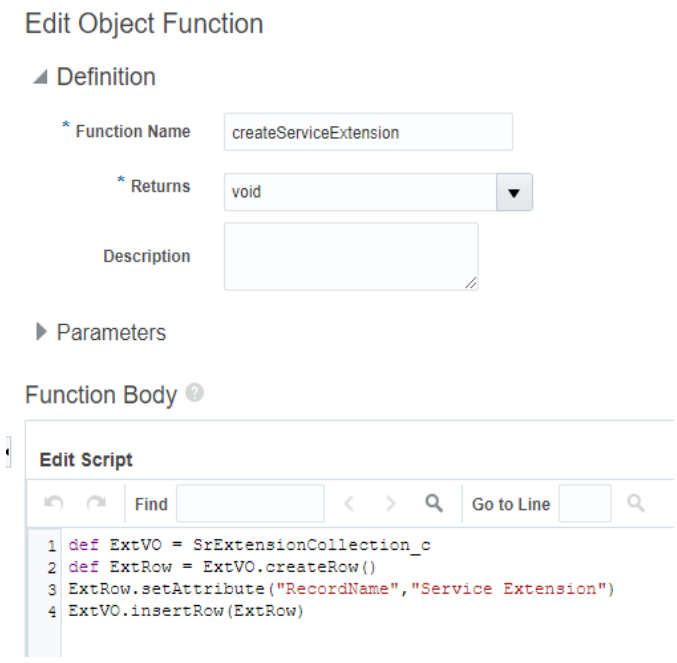

Figura 37. Função que instancia o objeto Service Extension

A página está dividida em diversas secções, que são apresentadas ou não consoante a categoria e/ou produto selecionados no *Service Request*. Por exemplo, para um Pedido de Serviço para solicitar uma recolha, a página apresenta as secções "Dados Gerais", a secção "Remetente/Destinatário" e a secção especifica do pedido, a "Recolha".

Além disso, existem secções comuns às várias categorias, uma delas é a do "Remetente/Destinatário", aquela onde estão incluídas as moradas e o "*Auto Complete*" das mesmas. Sendo que, o preenchimento automático mantém o mesmo funcionamento já explicado anteriormente (Figura 38).

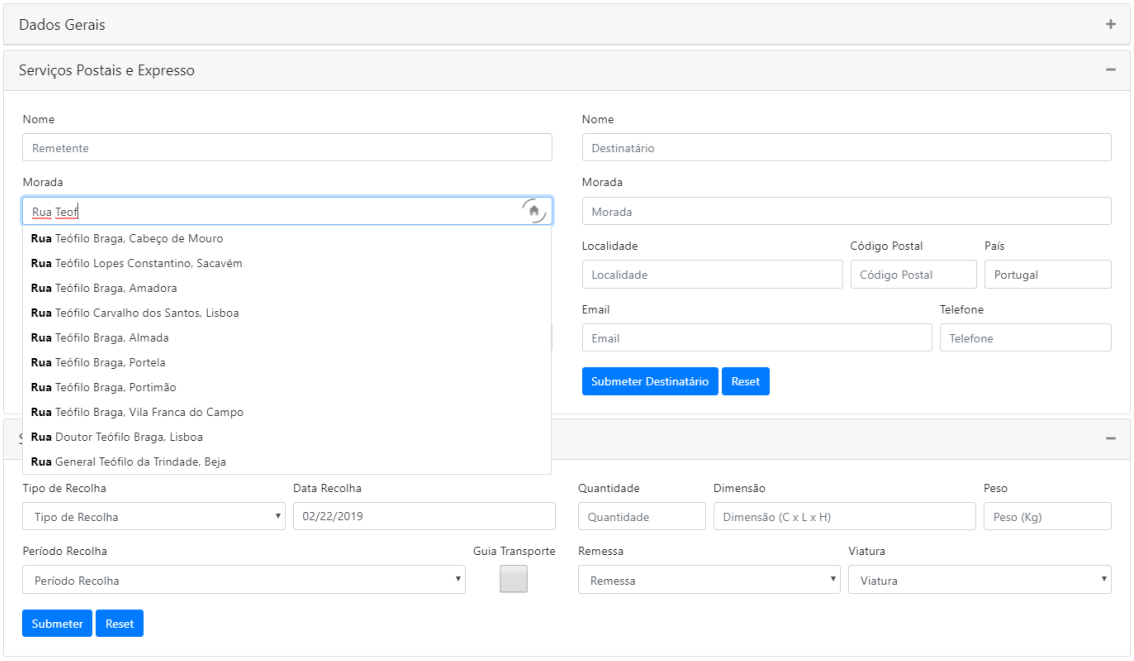

Figura 38. Excerto da visão da aplicação web dos dados complementares

A página é então integrada por "*Mashup*" numa aba lateral do *Service Request*, como é visível na imagem seguinte (Figura 39).

3.2.6 Conversão para *Java*

Como foi referido ao longo deste subcapítulo a aplicação externa, desenvolvida para os dados complementares, foi desenvolvida em *Javascript* e deste modo, também todas a chamadas e pedidos ao *OEC*. O problema do J*avascript* está na forma como é executado, ou melhor, o local onde é executado. Ao correr ao nível do *front end*, ou se quisermos, no *browser*, o *Javascript* vai expor os dados e credenciais que estejam a ser enviados nos pedidos. Inicialmente, isto não representava um problema, uma vez que, os dados enviados eram públicos e a autenticação era feita através de um *token* temporário. No entanto, e como já vimos, a aplicação cresceu para algo mais complexo e que envolve dados confidenciais e muitas mais chamadas a serviços externos do

cliente. Como tal, foi necessário reestruturar a aplicação para *Java*, sendo que a diferença está essencialmente na forma como os dados e os pedidos são feitos. Na nova versão todas as chamadas e pedidos são executados em *back end*, ou seja, ao nível da máquina servidora onde a aplicação está alojada, e não no browser de qualquer utilizador. A conversão passa por três fases principais, conversão das chamadas a *Web Services*, preparação de objetos de domínio sobre os quais se aplicam as possíveis regras de negócio e as ligações com o *front end*. Ao nível do *front end*, ou seja, a apresentação da aplicação, as alterações foram mínimas.

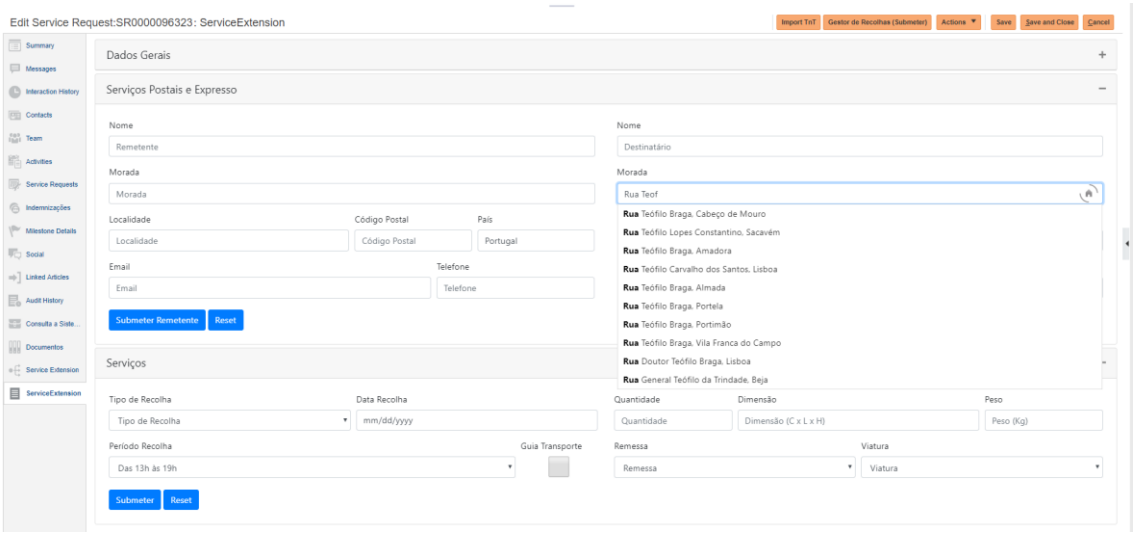

Figura 39. Excerto da visão da aplicação web dos dados complementares integrada no OEC no Service Request

A nova arquitetura segue um modelo *MVC* (*Model View Controller*), isto é, está separado em três segmentos primários, o *Model* que é a camada de acesso aos dados, neste caso é a camada que comunica com os serviços. A *View* que no fundo é a camada visível ao utilizador, é a camada que apresenta os dados e os envia para o *Controller*. O *Controller* é então a camada responsável pela comunicação entre a *View* e o *Model*. Por outras palavras o *Controller* recebe os dados da *View*, manipula-os e envia-os para o *Model*, e vice-versa. Para desenvolver este projeto *Java* as *frameworks* utilizadas foram o *Maven*, *Spring* e *Thymeleaf*.

De notar apenas que, este capítulo não pretende analisar exaustivamente o código produzido, mas sim a arquitetura, técnicas e ferramentas utilizadas para o desenvolvimento desta aplicação.

Ligação aos Serviços

A ligação com os serviços é feita, como já vimos, através de pedidos *REST* aos *Web Services* tanto do *OEC* como do cliente. Com a passagem para *Java* não é

diferente, ou seja, todos os serviços chamados são pedidos *REST*, mais especificamente pedidos *HTTP* ou *HTTP Requests*. Visto que os pedidos *REST* seguem sempre a mesma estrutura, *URL*, tipo, *headers*, etc. foi construída uma classe genérica para fazer qualquer tipo de pedido *REST*, um *REST Handler*. Depois disto, foi feita uma classe para cada ligação pretendida, por exemplo, ligação com o *OEC* continha dois pedidos, um do tipo *GET* e outro do tipo *PATCH*. Sendo que cada uma destas classes de ligação aos dados possui uma interface para comunicar com a camada seguinte. Estas classes recebem os parâmetros propagados das camadas acima e retornam, a essas mesmas camadas, os dados obtidos das chamadas aos *Web Services*.

Camada de Negócio

A camada de negócio é componente da aplicação responsável por obter, trabalhar e devolver os dados de acordo com as regras de negócio impostas pelo cliente. A este nível a aplicação foram feitas três divisões, a primeira para comunicar com a interface dos dados e dos serviços, uma componente intermédia para mapear e trabalhar os dados, e por fim a classe que comunica com a página visível ao utilizador.

A ligação à camada de dados não é feita invocando diretamente a sua classe, mas sim através da chamada a uma interface, como já referido. Esta chamada é feita nas classes *Service* que são no fundo as classes que invocam os pedidos e recebem os seus dados de resposta. As repostas vêm sempre na forma de um objeto *JSON*, o que não é o mais fácil de trabalhar a nível de regras de negócio. Como tal, em paralelo, existe uma classe para cada um dos objetos *JSON* das respostas. Ou seja, o objeto é mapeado para um objeto de domínio, com os atributos da classe iguais aos valores recebidos. Era também aqui que seriam aplicadas as regras, no entanto, estas foram mantidas ao nível do *OEC* e o seu funcionamento será explicado brevemente no próximo capítulo. Por fim, a classe *Controller* que obtém novamente os dados do objeto de domínio e os devolve para a página principal quando chamado.

Página para o Utilizador

A última camada, que no fundo é a primeira na linha de interação com o utilizador, é a responsável por exibir tanto os dados esperados como as suas regras de apresentação e obrigatoriedade. Esta camada sofreu poucas alterações em relação ao que já havia sido implementado, uma vez que as regras mais críticas do negócio foram de início implementadas dentro do *OEC*. Assim, a chamada dos dados mantém-se na forma de pedidos *HTTP* ou *Ajax* com *Javascript*, no entanto o *URL* invocado é um *URL* interno da aplicação e não o endereço direto do serviço e as suas credenciais. Esta chamada devolve novamente um objeto *JSON* com os dados, e aqui é um ponto em que questiono a arquitetura desta aplicação. Pois a devolução de um objeto *JSON* de novo torna

redundante grande parte da camada de negócio. No entanto, foi-me pedido que implementasse esta estrutura para que não destoasse de outra aplicação que foi também desenvolvida. Em seguida, e já com os dados, estes são distribuídos pelos respetivos campos utilizando o *Thymeleaf*. Esta ferramenta permite que, apenas com simples *tags* injetados diretamente no *html*, consigamos chamar os dados recebidos pelo seu nome e diretamente para o sítio pretendido, e ainda esconder ou exibir os campos pretendidos.

3.2.7 Regras e Validações de Campos

No *OEC* as validações dos campos são feitas através de pequenos scripts aplicados diretamente sobre o campo. Estes scripts consistem geralmente em simples condições e verificações que caso se confirmem verdadeiras produzem a regra. Dentro das regras podemos ter três tipos, regras de visibilidade, obrigatoriedade e atualização. Estas representam, respetivamente, se o campo está ou não visível, é ou não obrigatório e se pode ou não ser atualizado. Esta é a forma direta de fazer as regras, no entanto, é possível aplicar estas regras através de *scritps* disparados em *triggers*. Esta alternativa permite agrupar numa só mensagem todos as validações que falhem e devolver essa

```
//Mensagem de Erro
def errorMsg = "";

//Get SrExtension
SrExtensionCollection_c.reset()
if(SrExtensionCollection_c.hasNext()){
    def Extension = SrExtensionCollection_c.next()
    def CatLevelSC1 = getCatLevelSC(1);
    def CatLevelSC2 = getCatLevelSC(2);

    //Incidências Operacionais [GENERAL]
    if(CatLevelSC1 == 'INCID_OP'){
        if( Extension.Action_c == null ||
            Extension.Action_c == "" ){
            errorMsg += " Ação,";
        }
        if(Extension.Action_c == 'FREE_TEXT'){
          if(Extension.FreeTextArea_c == null ||
             Extension.FreeTextArea_c == ""){
            errorMsg += " Texto Livre,";
          }
        }
    }
    //Pedidos Gerais [GENERAL]
    if(CatLevelSC1 == 'PED_GER'){
        //Burlas
        if(CatLevelSC2 == 'CO_20005' ||
           CatLevelSC2 == 'EX_20055' ||
           CatLevelSC2 == 'RR_20002'){
```

Figura 40. Excerto do código de validação dos campos obrigatórios

mesma mensagem tanto para o *OEC* como para a aplicação externa no memento em que atualize os campos. Relembrando que a aplicação submete os dados para o *OEC* para que sejam guardados e neste momento é possível aplicar um *trigger* "*Before Update*" que irá validar os campos enviados e aqueles em falta (Figura 40).

3.3 *Oracle Business Intelligence*

Até aqui falámos sobre o trajeto dos dados. Ou seja, como os dados entram no sistema, como são geridos e tratados, quais as suas regras e como são armazenados no *OEC*. Neste capítulo, abordaremos a forma como podemos analisar e filtrar esses mesmos dados.

Um dos requisitos do cliente para este sistema *CRM* era ter uma ferramenta de análise e produção de relatórios dinâmicos sobre todos os dados armazenados no *OEC*. Para isso existe, associado ao *Engagement Cloud*, um sistema de análise de dados e produção de relatórios dinâmicos, o *Oracle Business Intelligence (OBI)*. Este sistema de *BI* é em tudo semelhante a outros sistemas conhecidos, como o *Power BI* ou o *QlikView*. Neste caso, como o *Oracle BI* está associado ao *Engagement Cloud*, torna-se mais simples pois as dimensões correspondem aos objetos dentro do próprio sistema *CRM*.

3.3.1 *Reporting* e *Analytics*

Uma vez que existe uma correspondência direta entre as dimensões no *OBI* e os objetos no *OEC*, torna-se simples gerar relatórios estáticos e dinâmicos referentes aos dados registados em sistema.

É possível definir áreas de análise, ou seja, uma vista sobre os objetos e dados que pretendemos analisar. Nessa área selecionamos quais os objetos e os respetivos campos onde estão contidos os dados para análise (Figuras 41 a 45).

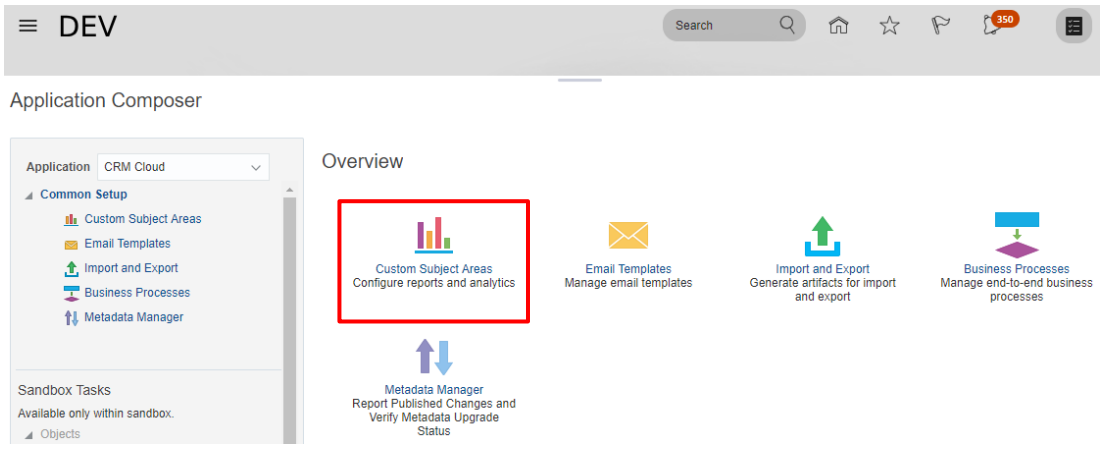

Figura 41. Visão do Application Composer, no menu overview, com destaque sobre a Opção de costumização de Subject Areas

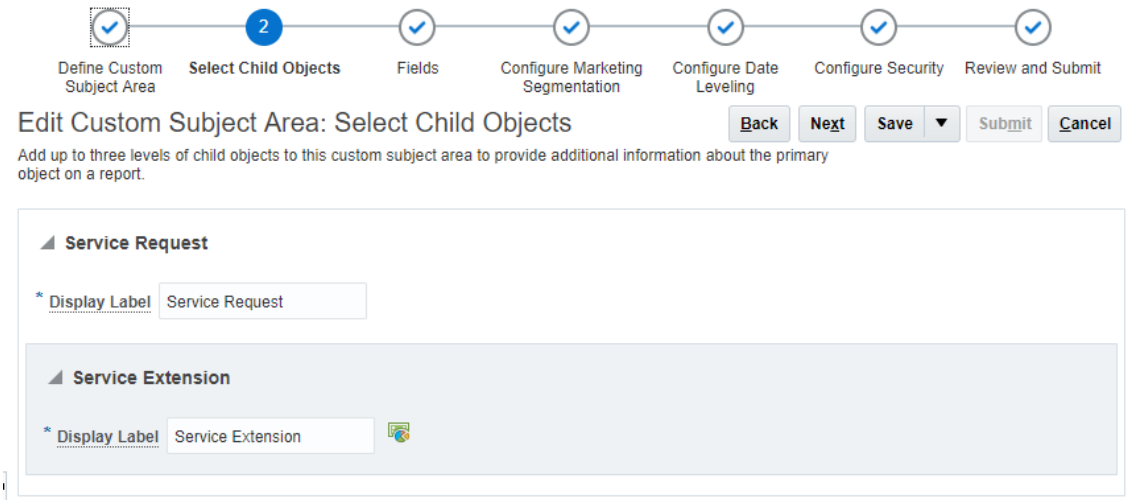

Figura 42. Menu de Criação/Edição de Custom Subject Areas (Etapa 2)

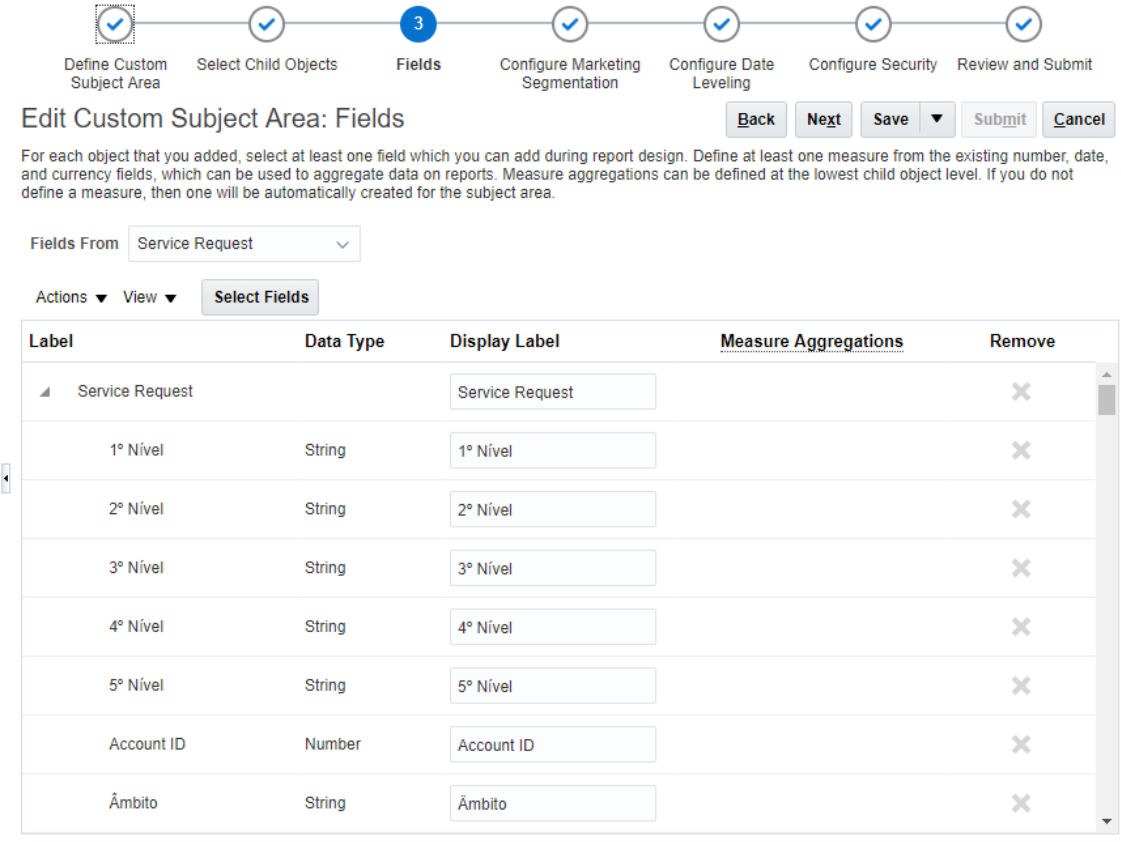

Figura 43. Menu de Criação/Edição de Custom Subject Areas (Etapa 3)

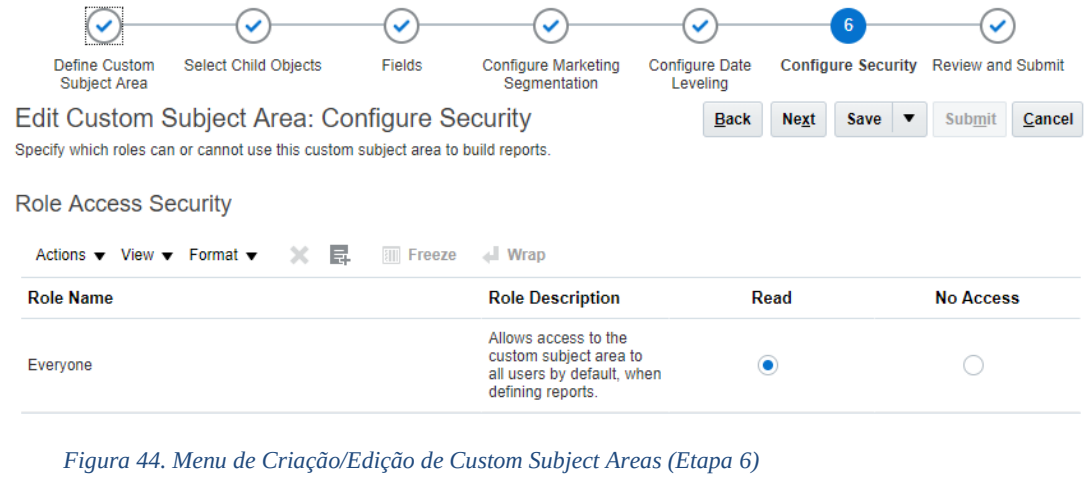

Figura 44. Menu de Criação/Edição de Custom Subject Areas (Etapa 6)

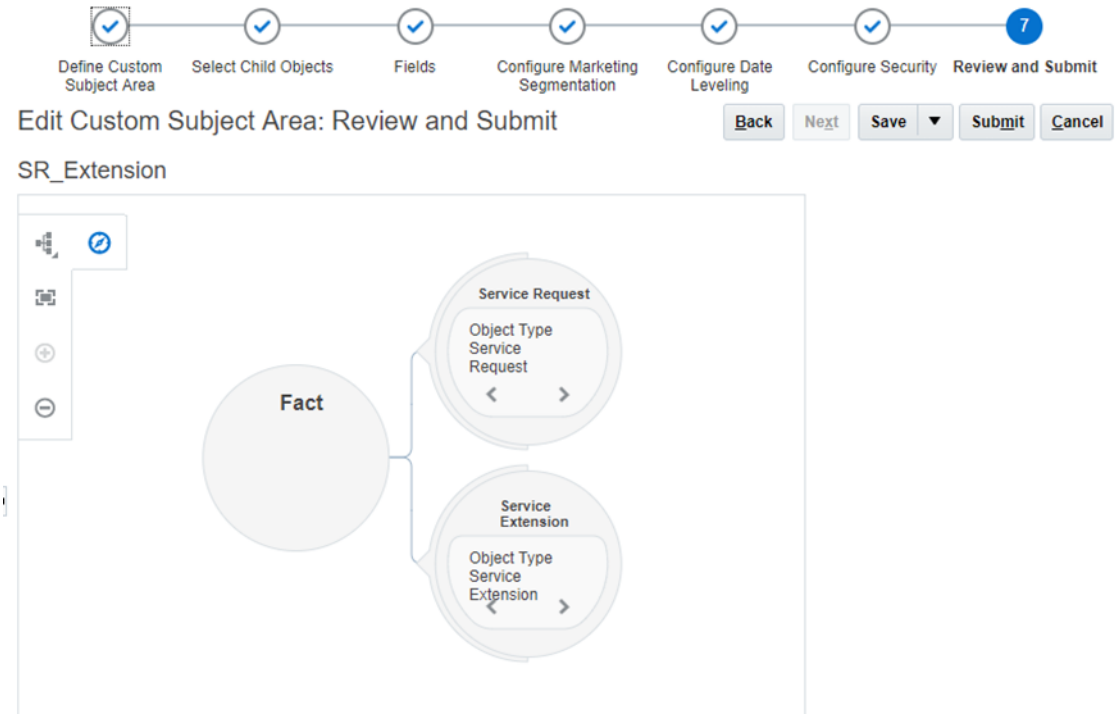

Figura 45. Menu de Criação/Edição de Custom Subject Areas (Etapa 7)

Após definida a área de análise torna-se possível, no *Oracle Business Intelligence*, criar uma nova análise de dados ("*New Analysis*") com esse mesmo grupo de dados.

Por exemplo, queremos realizar um relatório sobre quantos "*Service Requests*" com a categoria "Pedido Serviço" foram abertos, separados por âmbito (Nacional e Internacional) e por utilizador que os abriu.

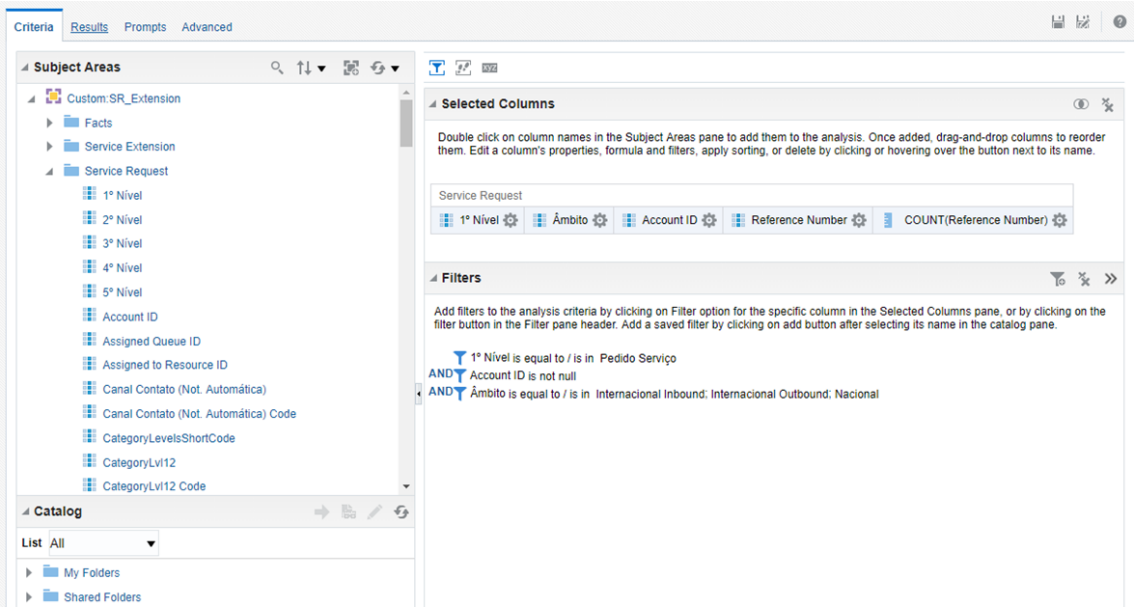

Figura 46. Visão sobre o ecrã de análise dos objetos Service Reuqest e Service Extension com os critérios e filtros de análise

Criada a nova análise no *B.I.* com base na área de análise definida, com recurso aos objetos *CRM* "*Service Request*" e "*Service Extension*", encontramos uma página com três janelas principais. A primeira janela é a de "*Criteria*", isto é, o local onde são selecionados os campos dos objetos que pretendemos averiguar e quais os filtros que pretendemos aplicar sobre estes (Figura 46).

No caso exemplo que estamos a seguir, desejamos saber quantos "*Service Requests*" com a categoria "Pedido de Serviço" foram abertos, separados por âmbito (Nacional e Internacional) e por utilizador que os abriu. Primeiro, temos que incluir o campo "1º Nível" uma vez que a categoria "Pedido de Serviço" é uma categoria de primeiro nível hierárquico, e uma vez que apena desejamos "Pedidos de Serviço" filtramos este campo por "Pedido de Serviço". Depois disto, incluímos os campos âmbito e conta visto que queremos separar os resultados precisamente por âmbito e por conta. Adicionamos também um filtro a ambos para evitar valores nulos. Por fim, arrastamos também o campo "*Reference Number*", ou seja, um identificador do "*Service Request*" apenas para lhe aplicar uma métrica sobre ele. Neste caso, a métrica é simplesmente a contagem das entradas distintas do campo "*Reference Number*".

Em seguida, passamos para a janela "*Results*" para consultar e ajustar os resultados dos critérios aplicados (Figura 47). Aqui é possível escolher diversos tipos de representações de dados, neste caso aplicamos uma "*Pivot Table*" onde podemos consultar quantos "*Service Requests*" foram abertos por conta tanto no âmbito nacional como internacional.

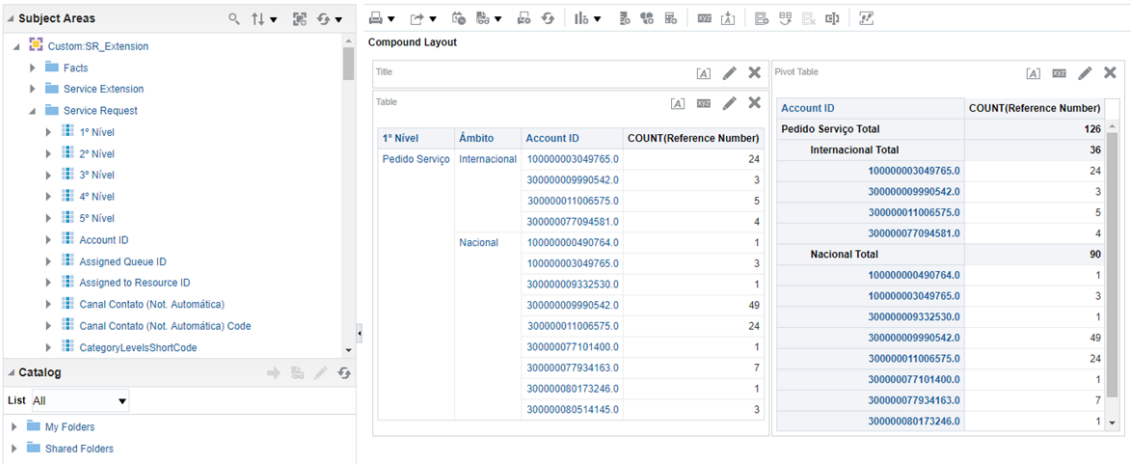

Figura 47. Visão sobre os resutlados da análise

Por fim, temos a janela de "*Prompts*" que é nada mais nada menos que o local onde podemos definir caixas de seleção múltipla para os campos que incluímos na nossa análise. Neste caso, definimos uma caixa de seleção para os valores do âmbito, "Nacional" e "Internacional" (Figura 48).

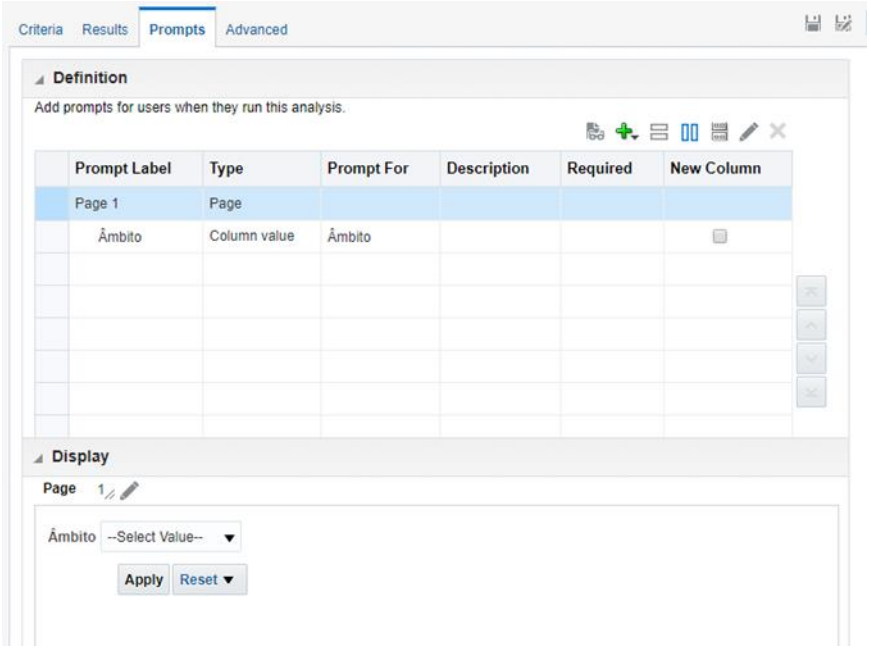

Figura 48. Visão sobre o menu de Prompts

Assim, o utilizador pode filtrar ele próprio os resultados com base no âmbito. O utilizador, no entanto, irá aceder a estes relatórios diretamente no *Oracle B.I.* (apesar de o poder fazer), mas sim no próprio *OEC* onde os relatórios podem ser embebidos e dispostos.

3.4 Análise de Fluxos e Desenho de *Use Cases*

A análise de requisitos é uma componente essencial do desenvolvimento de um projeto. Os requisitos do cliente são traduzidos para fluxos de execução e ficheiros descritivos das funcionalidades e variáveis necessárias. Deste modo, por forma a implementar uma tarefa/requisito é necessário analisar os documentos que a descrevem, sendo que por vezes é ainda necessário discutir ajustes diretamente com o cliente. Dado isto, cada tarefa cumprida deve ser também registada, ou seja, documentada num ficheiro com todas as implementações feitas até ao momento. Esse ficheiro serve de guia de implementação, para as eventuais reconstruções do ambiente.

3.4.1 *Demo* dos Pedidos de Serviço - Recolhas

A terceira demonstração do produto ao cliente, serviu para apresentar os fluxos de criação, alteração e cancelamento de um Pedido de Serviço - Recolha (Categoria que define uma recolha). Para isto foi necessário analisar os fluxos, ou seja, os grafos sequenciais que descrevem as operações envolvidas nos processos de criação, alteração e cancelamento de um Recolha, e desenhar os casos de uso para apresentar. Os casos de uso começam por criar um "*Service Request*" com a categoria "Pedido de Serviço - Recolha". Em seguida, aceder á página de preenchimento de dados extra (a página do *Service Extension*) e preencher os dados necessários para solicitar uma recolha, inclusive as moradas de origem e destino, a data e horário, etc. Em seguida, submeter estes dados para um gestor externo, desenvolvido pela equipa de integração. Daqui em diante, o *OEC* apenas notifica o cliente com o estado atual do pedido.

Para o caso dos pedidos de alteração/cancelamento, o fluxo é semelhante, no entanto, para solicitar a alteração ao pedido de recolha, é necessário criar um novo "*Service Request*", com a categoria "Pedido de Serviço - Alteração/Cancelamento Recolha", efetuar as alterações desejadas e submeter novamente esse pedido. Sendo que esse pedido inclui o número associado ao pedido de recolha original, para que possa ser feita a mudança. Os passos genéricos dos casos de uso apresentados estão representados no slide em baixo (Figura 49).

No.	Cenário	Key Role(s)	O que pretendemos demonstrar?
1	**Chamada Inbound** Reincidência de uma Solicitação, com Reabertura	CC- Agent (TestUser2) / Back Office – Andreia Lopes 	1. TestUser2 Regista Solicitação e confirma que se trata de uma Reincidência 2. Associa solicitação "nova" a solicitação fechada e Encaminha 3. Andreia inicia tratamento da solicitação e decide Reabrir a Solicitação anterior 4. Fecha a Solicitação criada pelo TestUser2
3	**Chamada Inbound** Pedido de Serviço > Recolha Âmbito Internacional Outbound	Correios BU - Agent (TestUser2) 	1. Registo manual do Pedido de Serviço > Recolha 2. Preencher os campos da Recolha e das Moradas 3. Submeter os dados para o GESTOR DE RECOLHAS (GR) -> Pendente Integração 4. Receber aprovação do GR -> Em Progresso > Aprovado e notifica cliente por SMS/EMAIL? 5. Estado volta para Pendente Integração (GR) 6. GR Resolve o Pedido de Serviço e altera o Estado para Resolvido e Efetuado 7. OEC envia a notificação para o cliente
2	**Email Inbound** Pedido de Serviço > Alteração e Cancelamento de Recolha	Correios BU - Agent (TestUser2) 	1. Replicar procedimento do cenário anterior (1 – 5) 2. Cria um novo Pedido de Serviço > Alteração Recolha 3. Adiciona o número da recolha e efectua as alterações ao pedido -> Submeter para GR 4. Receber aprovação do GR -> Fechar pedido de alteração de Recolha e voltar ao original 5. Notificar o cliente da Alteração do Pedido de Recolha por SMS/EMAIL? 6. (? EDIDO DE CANCELAMENTO ?) 7. GR fecha pedido de Reoclha com estado Resolvido e Efectuado 8. OEC notifica cliente

Figura 49. Conjunto de Casos de uso desenhados para a demonstração das implementações ao cliente

3.5 Preparação dos Testes Automáticos

A fase de testes de uma aplicação é sempre uma tarefa exaustiva e morosa, pelo que exige sempre um investimento de recursos humanos e tempo. Como tal, e com vista a automatizar processos dentro do *OEC* foi pensada uma solução para libertar tempo e esforço durante as fases de teste da aplicação.

Como foi referido anteriormente neste documento, a fase de testes deste projeto inclui a fase de testes de integração e a fase de testes de aceitação. Estas etapas foram executadas manualmente e esta automação não se destinava a eles. Os testes mais críticos e que levaram ao planeamento desta solução foram os testes *pós-upgrade*, ou seja, testes que devem ser realizados sempre que a aplicação sofre uma atualização. Uma vez que o *Engagement Cloud* é relativamente recente, o número de atualizações e alterações ao produto são regulares e visíveis. Perante isso, é necessário executar novamente os testes ao ambiente por forma a garantir que o seu funcionamento não foi afetado. E é então aqui que os testes automáticos iriam atuar, tanto para a componente de *Sales* como para a componente, em presente desenvolvimento, do *Service*.

Visto que estamos a considerar dois setores inteiros da aplicação, ainda para mais, altamente customizados, a lista de testes torna-se extensa, mas no fundo, os testes são os mesmos realizados nas fases de integração e aceitação. Estes testes podem ser mais ou menos complexos, e passam pela execução de ações e fluxos com objetivos específicos. Essas ações são, por exemplo: "Fazer *Login*", "Abrir o Separador dos Pedidos de Serviço (*Service Requests*)", "Criar um novo Pedido de Serviço", entre outras.

Dentro de cada um desses passos está naturalmente implícito o preenchimento e validação de campos, o clique em botões, etc.

Para automatizar tudo isto foi pensada a utilização de uma aplicação em *Java* construída com as *frameworks Cucumber*, *Selenium*, *Junit* e *Maven*. Estas *frameworks* estão aplicadas na definição, criação e execução dos testes.

A opção do *Maven* tem um propósito organizacional, ou seja, facilita a organização e atualização, sempre que necessária, de todas as dependências do projeto. Assim, foi criado um projeto *Maven* cujo ficheiro "*pom.xml*" contém todos os *plugins* e dependências requeridos para a execução do projeto (Figura 50). O projeto tem o conteúdo dividido por diferentes pastas e *packages*, com classes *Java* e outros ficheiros essenciais para o seu funcionamento (Figura 51).

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.53.0</version>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-server</artifactId>
    <version>2.53.0</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.3</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-picocontainer</artifactId>
    <version>1.2.3</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.2.3</version>
    <scope>test</scope>
</dependency>
```

Figura 50. Ficheiro pom.xml de configuração do projeto de automação de testes

Nas pastas *src/test/java* e *src/test/resource* estão os ficheiros principais para criar e executar um cenário de teste automático. Esta estrutura é aplicada devido ao *Maven.*

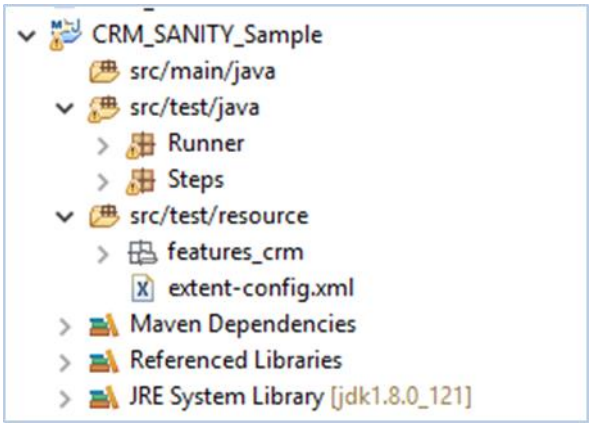

Figura 51. Organização das pastas no projeto de testes automáticos

Em seguida será apresentada a construção de um cenário de teste, ilustrativo das capacidades e vantagens desta abordagem automática. Para isso começamos por definir quais os passos (*Steps*) que fazem parte do teste automático. Através da sintaxe *Gherkin* do *Cucumber* podemos definir o cenário e os passos a serem executados, nomeando cada passo do teste com frases simples e que definam as ações a serem executadas (como de pode ver na Figura 52, abaixo). Este código deve ser escrito num ficheiro "*.feature*" num *package* da pasta *src/test/resource*.

```
1  Feature: Titulo1
2
3    @TAG
4    Scenario: TESTE1
5      Given Abrir IE
6      And Aceder ao OEC
7      When Fazer login
8      And Carregar HomePage
9      And Aceder a service
10     And Aceder a service request
11
```

Figura 52. Código Gherkin, no Cucumber que define um teste exemplo básico de login e acesso aos pedidos de serviço

As explicações seguintes terão por base um exemplo simples de testes automáticos, onde o cenário de teste será o seguinte:

1. Abrir o *browser*
2. Aceder ao *Oracle Engagement Cloud*
3. Fazer o *Login* com as credenciais válidas
4. Abrir a página inicial do *OEC*
5. Aceder à secção do *Service*
6. Abrir os *Service Requests*

Por outras palavras, o que este teste vai fazer é abrir o browser, abrir e fazer login no *OEC*, e aceder à secção "*Service*" e de seguida "*Service Requests*".

Cada cenário tem uma "*tag*" ou nome específico que o identifica, sendo que este nome é relevante pois será depois usado na escolha dos casos a executar. Esta *tag* é composta por um "@" seguido do nome desejado, no caso apresentado acima temos "*@TAG*".

Uma vez definidas as regras para o cenário de teste, no ficheiro "*.feature*", é necessário implementar uma classe *Java*, dentro de uma package na pasta *src/test/java*. Neste caso o ficheiro *Java* tem o nome "*TAG.java*", pois este ficheiro diz respeito ao cenário de teste. Neste ficheiro, são definidos métodos para cada um dos passos descritos para um cenário, iniciando o método com a *tag* reservado do *Cucumber*, por exemplo "*@Given("Abrir Internet Explorer$")*". Na figura seguinte (Figura 53) pode observar-se a construção dos métodos que implementam cada passo do cenário definido na figura anterior (Figura 52), bem como a utilização da classe "*Webdriver*" pertencente à biblioteca *Selenium*.

```
@Given("^Abrir IE$")
public void abrir_IE() throws Throwable {
    driver = shared.abrir_browser();
    wait = new WebDriverWait(driver, 30);
}

@Given("^Aceder ao OEC$")
public void aceder_ao_OEC() throws Throwable {
   driver.get("https://cdat-test.login.em3.oraclecloud.com");
}

@When("^Fazer login$")
public void fazer_login() throws Throwable {
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("userid")));
    driver.findElement(By.id("userid")).sendKeys("USER");
    driver.findElement(By.id("password")).sendKeys("PASSWORD");
    driver.findElement(By.id("btnActive")).click();
}

@When("^Carregar HomePage$")
public void carregar_homepage() throws Throwable {

    Thread.sleep(5000);
    driver.findElement(By.id("pt1:_UIShome")).click();
}

@When("^Aceder a service$")
public void aceder_a_service() throws Throwable {
    shared.Screenshot(scenario);
    driver.findElement(By.xpath("//div[@id='groupNode_service']")).click();
}

@When("^Aceder a service request$")
public void aceder_a_service_request() throws Throwable {
    driver.findElement(By.xpath("//div[@id='itemNode_service_service_request']")).click();
}
```

Figura 53. Métodos que configuram os passos do cenário definido para o teste automático exemplo

Por fim, no *package* "*Runner*" incluímos a classe "*RunnerTest.java*" que é a classe que irá executar o projeto. Nesta classe definem-se também, quais as *tags* dos testes que se pretende executar, bastando para isso introduzir os nomes na lista de *tags*, como está visível na imagem seguinte (Figura 54).

```
@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"com.cucumber.listener.ExtentCucumberFormatter:output/report.html",
        "junit:target/cucumber-results.xml"},
    features = {"src/test/resource"},
    glue = {"Steps"},
    tags = {"@TAG"},
    format = {"pretty","json:target/json", "html:target/html-exemplo"+"TestSemPrinte"})

public class RunnerTest extends  ExtentCucumberFormatter{

    @BeforeClass
    public static void setup() {
        // Initiates the extent report and generates the output in the output/Run_/report.html file by default.
        ExtentCucumberFormatter.initiateExtentCucumberFormatter();

        // Loads the extent config xml to customize on the report.
        ExtentCucumberFormatter.loadConfig(new File("src/test/resource/extent-config.xml"));
```

Figura 54. RunnerTest.java, classe que executa os vários cenários de teste

Com a implementação completa, executa-se o projeto como um teste *Junit*, a partir da classe "*RunnerTest.java*" previamente mencionada (Figura 55). Os testes são executados por ordem de introdução no campo *tags* na classe "*RunnerTest.java*". Assim sendo, cada cenário de teste tem os seus cenários e passos testados por ordem sequencial. O *browser* escolhido na implementação, neste caso do exemplo o *Internet Explorer*, abre e executa as ações definidas automaticamente.

Relativamente à *API Webdriver*, é uma *API* que nativamente executa ações nos browsers suportados. Esta *API* permite identificar os elementos presentes nas páginas *web* de diferentes formas, quer pelos *IDs* dos elementos quer por *Xpath*. Sendo que é preferível a utilização via *ID* pois geralmente são identificadores únicos e muito mais rápidos. Por exemplo:

driver.findElement(By.id('login_user'));

Por outro lado, com o *Xpath* encontramos uma grande variedade de escolha na identificação de elementos pois é possível identificar por qualquer atributo do elemento como por exemplo *id*, *class*, *value*, *type*, *src*, *style*, *key*, etc., assim como pela própria estrutura do *html*. A título de exemplo:

driver.findElement(By.xpath("//a[contains(@id,'_vnsearchvar_variant-btn')]"));

driver.findElement(By.xpath("//a[@key='ZZBUILHEADER']"));

driver.findElement(By.xpath("//a[@title="Visão 360']"));

driver.findElement(By.xpath(".//[contains(@id,'btcategory_cat02')]/ul/li[2]/a"));*

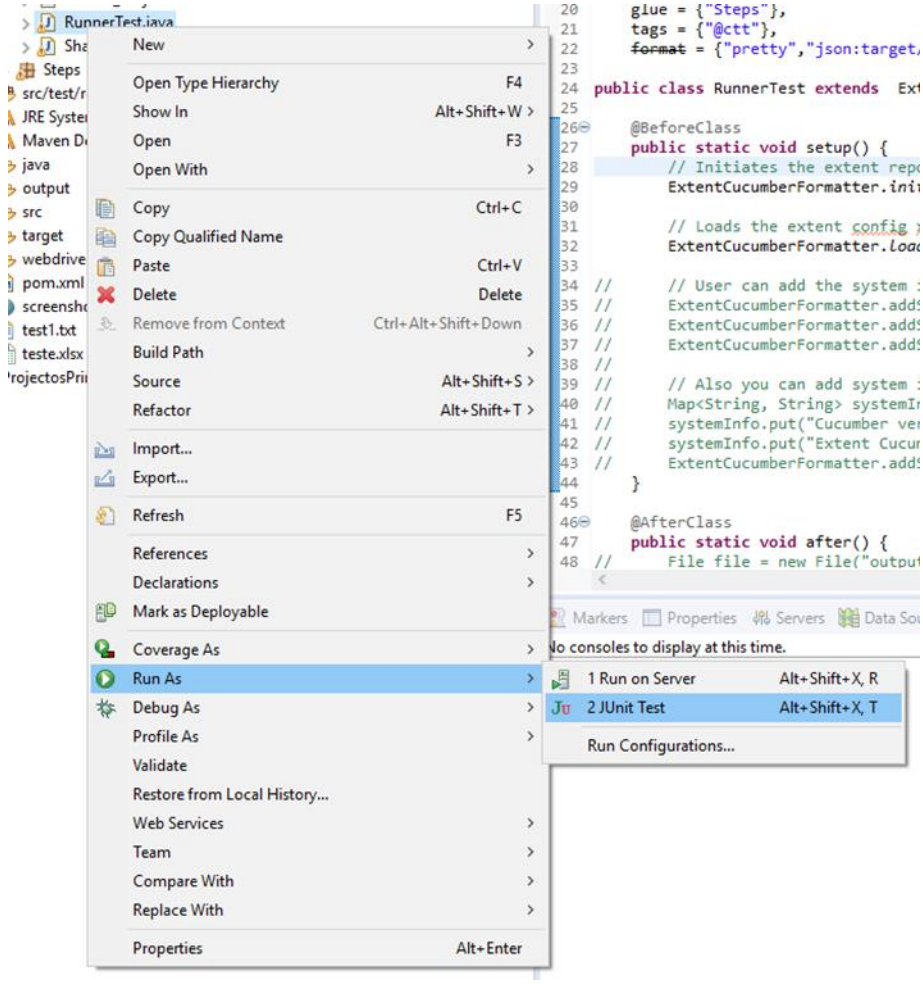

Figura 55. Execução do projeto de automação de testes exemplo com recurso ao Junit

Por último, em relação aos testes automáticos, é possível produzir relatórios de execução. Estes relatórios são apresentados no *browser* e incluem uma descrição com os resultados de cada passo do cenário de testes individualmente (Figura 56).

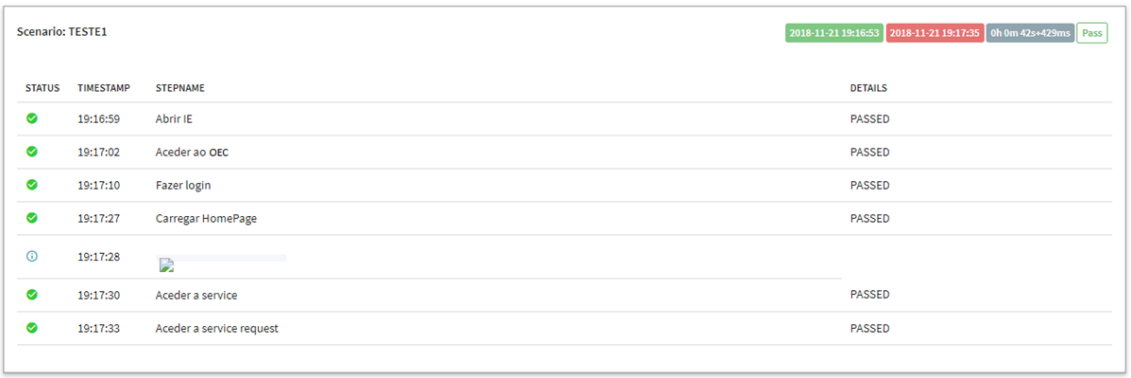

Figura 56. Página web com o relatório gerado após a execução do teste automático exemplo

Capítulo 4

Conclusões

4.1 Sumário

O objetivo deste projeto era desenvolver a componente *Service* de um *CRM* recorrendo ao *Oracle Engagement Cloud* para a implementação. O *Oracle Engagement Cloud* é um produto novo, mas só ainda mais recentemente se expressou em Portugal, sendo que este projeto que integrei é dos primeiros, se não mesmo o primeiro a ser implementado numa empresa portuguesa. Como tal, o início do projeto foi um momento de aprendizagem para toda a equipa. Ao trabalhar num projeto e com uma equipa tem-se também contacto com metodologias e estilos de trabalho. Metodologias essas que, apesar de não me serem desconhecidas, tomam todo um novo significado no contexto profissional. E esse será talvez um dos grandes marcos deste estágio, o contacto direto com o mundo profissional.

Neste projeto, o *CRM* apresenta-se na forma de aplicação *cloud*, e está dividido em *Sales* e *Service*, uma centra-se no cliente enquanto entidade e outra no cliente enquanto requisitante de serviços. O *Service*, que foi o nosso foco neste trabalho, centra-se muito em torno dos "Pedidos de Serviço" ou "*Service Requests*". Com isto entenda-se, a sua categorização e preenchimento com informações relevantes, gestão de ciclos de vida, e comunicação informativa para o cliente.

Chegando aqui, sobra sumariar o trabalho desenvolvido, relembrando que todo o trabalho enunciado ao longo deste relatório foi realizado diretamente por mim. Outros trabalhos em que participei, mas que não me foram atribuídos diretamente foram ignorados, como por exemplo a ajuda na integração do *OEC* com *SAP* ou a ajuda no desenvolvimento de uma funcionalidade de atualização massiva de *Service Requests* e subsequente envio de email para o cliente. Estes trabalhos, apesar de terem tido o meu contributo, não me foram atribuídos e acabam por ser semelhantes a outras tarefas referidas neste relatório.

O trabalho desenvolvido pode ser divido em três grandes componentes, o trabalho dentro do *OEC*, o trabalho fora do *OEC* e o trabalho no *Oracle Business Intelligence*.

O trabalho no *OEC* esteve em customizar a aplicação base e configurá-la de forma a cumprir os requisitos de negócio. Isto passa pela criação de campos e respetivas regras, pela criação de *triggers* e funções para eventos específicos. Tudo isto escrito na linguagem do sistema, o *Groovy*. Configuração das comunicações de entrada e de saída, ou seja, *emails* e *SMS*, e os respetivos *templates*. Envolve a integração de *Web Services* e *APIs* externas. Isto é feito configurando o *endpoint* desejado e o método *REST* que pretendemos invocar, e de seguida invocar esse *Web Service* numa função, *trigger* ou *workflow* onde o *JSON* devolvido possa ser tratado e utilizado. Configuração dos perfis de segurança, que por sua vez permitem a configuração e o acesso a outras áreas do sistema como objetos ou importação massiva dos mesmos. A importação de dados para a aplicação é outra das componentes mais relevantes, em particular a importação massiva das categorias. Sendo que, para importar as categorias, estas foram primeiro divididas em dez diferentes unidades de negócio, algo que também foi customizado. Sendo que uma unidade de negócio é, como o nome indica, uma divisão de muitos dos componentes da aplicação em setores definíveis. Outra das grandes implementações feitas foram as *Entitlement Rules*, sendo que esta é uma das configurações mais complexas ao nível do *OEC*. Estas regras servem, muito sucintamente, para delimitar temporalmente as principais etapas no ciclo de vida de um *Service Request*. Contabilizando o tempo desde o seu início até ao seu desfecho, bem como o tempo que resta para que ele seja resolvido. Por fim, abordaram-se também as regras de atribuição, que são no fundo a forma automática de atribuir um *Service Request* a um operador responsável por o acompanhar e resolver.

Apesar de permitir inúmeras customizações, nem sempre a aplicação permite ou é possível construir a visão exata do cliente. Para essas situações foi construída uma aplicação externa que complementasse as lacunas do *OEC*. Inicialmente começou por ser um simples *Autocomplete* de moradas. No entanto, rapidamente se passou para uma aplicação complexa com cerca de trezentos campos espalhados por várias abas, e regulados por regras de visibilidade, obrigatoriedade e atualização e ainda com integração com vários serviços. Por fim, foi feita uma conversão do código *Javscript* para uma aplicação *Java*, uma vez que, é assim garantida a segurança das credenciais bem como dos dados afetos a uma chamada *REST*. Tudo isto pelo simples facto de a aplicação passar a executar ao nível do servidor e não ao nível do browser do utilizador.

Por fim, o trabalho realizado no *Oracle BI* é um trabalho de *reporting* comum. Isto é, o trabalho de configuração dos dados e das condições pretendidas que levem á produção de tabelas e relatórios com a informação desejada pelo cliente. Essas tabelas e medidas associadas aos campos são aquilo que no final mais importa para quem pretende analisar o estado e valores da aplicação.

Fora das fases de desenho e desenvolvimento estão as fases de testes. Apesar de não ter realizado os testes diretamente, estive dentro da solução de automatização dos mesmos. Isto significou uma investigação e preparação da tecnologia de automatização de processos. Infelizmente o projeto não teve recursos para aplicar essa solução e, como tal, os testes de integração, aceitação e após atualizações, permaneceram totalmente manuais.

4.2 Discussão

O *Oracle Engagement Cloud* é uma tecnologia *cloud* muito recente e isso traz vantagens ao nível do mercado laboral, uma vez que, caso a tecnologia ganhe popularidade, quem foi pioneiro na sua utilização terá uma maior rentabilidade no mercado. No entanto, esta breve existência do *OEC* acarreta também alguns problemas. Um deles, é a quantidade de *bugs* que naturalmente ainda existem, quase sempre contornáveis, mas que exigiram uma constante comunicação com as equipas de suporte da *Oracle*. Outra desvantagem está na sua aceitação no mercado, uma vez que sendo uma tecnologia recente, nem todos os investidores e empresas têm a confiança de apostar nela. Por outro lado, ainda é necessário existir uma consciência da complexidade dos projetos sobre os quais pretendemos aplicar o *OEC*. O *OEC*, à data da redação deste documento, não aparenta estar desenhado para uma complexidade de negócio tão elevada como a que foi aqui planeada. Uma dessas situações está visível com o desenvolvimento da página externa para dados complementares. Caso tivéssemos optado por utilizar o *OEC* na sua totalidade teriam que ser criados inúmeros *layouts* (páginas) e condições nos *Service Requests* para que a apresentação dos diversos campos e secções fosse dinâmica. Esta complexidade levou-nos então, entre outros motivos, a esta página externa para os campos e regras, sendo que, com essa página, o *OEC* passou a ter um papel de base de dados, pois apesar de incluir também as regras de negócio, a sua grande função era a de armazenar os dados dos campos. Isto acaba, infelizmente, por descreditar um pouco o *OEC*, não obstante, a integração com aplicações e serviços externos é limpa e totalmente funcional. Ainda acerca da complexidade do negócio implementado é importante relembrar que, o *CRM* não é apenas uma transposição do negócio para um bloco de software, o *CRM* é acima de tudo uma reinvenção do negócio para que se centre no cliente. Neste caso a complexidade de negócio deveria ter sido um pouco mais repensada por forma a evitar um volume de carga desnecessário o sistema e para quem o utiliza.

Relativamente a trabalho futuro para o projeto, fica ainda pendente a integração com os restantes serviços legados do cliente. Isto é, invocar e tratar todos os *Web Services* e *APIs* em falta, tanto diretamente pelo *OEC* como pelas aplicações customizadas, em especial a aplicação dos “Dados Complementares”, desenvolvida na

íntegra por mim. No fundo as restantes alterações e desenvolvimentos que dizem respeito aos *Sprints* 2 e 3. Fica também pendente a fase de testes pós passagem para o ambiente de qualidade e respetiva correção de *bugs*. Isto significa, naturalmente, que fica também por fazer a passagem para o ambiente final de produção. Por fim, é ainda espectável a implementação de mais módulos dos disponíveis no conjunto da *Oracle*. Dentro destes, está já assegurada a implementação de pelo menos mais um dos módulos, a componente de *Marketing*.

Bibliografia

[1] *Customer Relationship Management: Concepts and Tools*, Francis Buttle, 2004

[2] https://www.propellercrm.com/blog/leads-vs-opportunities : Último acesso em 17 de julho de 2019.

[3] *Engagement Cloud HR Help Desk: Sales, Oracle*

[4] *Engagement Cloud HR Help Desk: Service, Oracle*

[5] https://www.tutorialspoint.com/groovy : Último acesso em 17 de julho de 2019.

[6] https://jwt.io/introduction : Último acesso em 17 de julho de 2019.

[7] "*Magic Quadrant for CRM and Customer Experience Implementation Services, Worldwide*" *by* Patrick J. Sullivan *and* Ed Thompson. 8 *January* 2018

[8] http://scrummethodology.com : Último acesso em 17 de julho de 2019.

[9] https://stackify.com/gradle-vs-maven : Último acesso em 8 de agosto de 2019